09784674.021501

```
Attribute VB_Name = "General"
Option Explicit

'Program and DB Rev
Public Const ProgRev$ = "PP Rev 1.00"
Public Const DBRev$ = "PP MDB Template 1.04"

'Columns collection
Public PSColumns As New Collection

'Statistics XArray
Public StatArray As New XArray

'Registry keys
Public Const SKEY_PPPPP$ = "PamelaProbeDesigner"       'application key
Public Const VALUE_DB_TEMPLATE$ = "DBTemplate"         'workspace DB template
Public Const VALUE_DB_DIR$ = "DBDir"                   'workspace DBs directory
Public Const VALUE_DB_FILE_EXT$ = "DBFileExt"          'workspace DB extension
Public Const VALUE_GB_DIR$ = "GBDir"                   'Genbank directory
Public Const VALUE_GB_FILE_EXT$ = "GBFileExt"          'GenBank file extension
Public Const VALUE_BLAST_DIR$ = "BLASTDir"             'BLAST databases
directory

'Error codes
Enum ppErrors
    ppErrGBFileLength = 3000
    ppErrGBFileFormat = 3001
    ppErrBadRecordset = 3002
    ppErrNoProbesCreated = 3003
    ppErrCancelled = 3004
    ppErrDBFormat = 3005
    ppErrBlastVersion = 3006
    ppErrFASTAFormat = 3007
    ppErrSeqLoadDB = 3008
    ppErrDLL = 3009
End Enum

'Flags to prevent sequence and probeset selectors firing.
'Currently broken -- RowColChange events are ignored.
Public AllowdbgClick As Boolean

'Algorithm parameter sets
Public CreatePSPars As New cCreatePSPars
Public PosFilterPars As New cPosFilterPars
Public LengthFilterPars As New cLengthFilterPars
Public GCFilterPars As New cGCFilterPars
Public dGDPars As New cdGDPars
Public dGDFilterPars As New cdGDFilterPars
Public TMPars As New cTMPars
Public TMFilterPars As New cTMFilterPars
Public dGHPars As New cDGHPars
Public dGHFilterPars As New cdGHFilterPars
Public dGMPars As New cdGMPars
Public dGMFilterPars As New cdGMFilterPars
Public RunPars As New cRunPars
Public RunFilterPars As New CRunFilterPars
Public Progress As New cProgress
Public EntrezPars As New cEntrezPars
```

```
Public TrimPars As New cTrimPars
Public ClampPars As New cClampPars
Public ClampFilterPars As New cClampFilterPars
Public BlastPars As New cBLASTPars
Public HomologyPars As New cHomologyPars
Public HomoFilterPars As New cHomoFilterPars
Public ArrayPars As New cArrayPars

' SeqRecord Structure
' This structure holds read from one sequence file/DB/website.
' It it used when reading Genbank files (in which case all fields
' are filled in), or FASTA files (in which case only the Header and
' Sequence portions are filled in).
Public Type SeqRecord
    Header As String
    Locus As String
    Accession As String
    Length As Long
    Sequence As String
End Type

Sub Main()

'Initialize thermodynamic parameters.
InitThermoPars

'Start it up!
frmMain.Show

End Sub
```

```
Attribute VB_Name = "Blast"
Option Explicit

Private Declare Function InitPH Lib "homologydll.dll" _
    (ByVal DB$, ByVal NumProbes&, ByVal W&, ByVal MaxProbeLength&) As Long

Private Declare Function AddProbe Lib "homologydll.dll" _
    (ByVal Probe$, ByVal ProbeLength&) As Long

Private Declare Function CalcOneHomology Lib "homologydll.dll" (ByVal
SeqLength&) As Long

Private Declare Function SkipSequence Lib "homologydll.dll" (ByVal SeqLength&)
As Long

Private Declare Function GetHomology Lib "homologydll.dll" (ByRef Homology&) As
Long

Private Declare Function TerminatePH Lib "homologydll.dll" () As Long

Public Const MaxBLASTHeaders& = 100
```

09284674.021501

```
Public Sub CalcHomology(Seq$(), HP As cHomologyPars, Homology&(), RSBlast As
Recordset, PSName$)
'---------------------------------------------------------------------------
'Function
'   Calculate the homology of an array of probes to a database of sequences.
'Arguments
'   Seq: The sequences.
'   HP: The homology parameters.
'   Homology: The homologies.
'   RSBLast: The table of BLAST-located homologies.
'   PSName: The name of the probeset we are processing.
'---------------------------------------------------------------------------
On Error GoTo E

Dim NumSeqs&          'number of sequences we are working with
Dim MaxProbeLen&      'longest probe
Dim P&, S#            'index
Dim DBFile&           'header file for database
Dim SR As SeqRecord   'one header from the header file
Dim LineText$         'one line from the header file
Dim NumHeaders&       'number of headers in the header file
Dim DoHomology As Boolean       'is homology to be done for this sequence
Dim foo&              'bit bucket
Dim N&, NumNoCheck&
Dim NoCheck$()        'list of accession numbers not to check homology against
Dim HomologyBaseName            'base name of the DB files needed for homology

'Determine the number of probes we are calculating homology for.
NumSeqs = UBound(Seq) + 1

'Find the longest probe.
MaxProbeLen = 0
For P = 0 To NumSeqs - 1
    If Len(Seq(P)) > MaxProbeLen Then MaxProbeLen = Len(Seq(P))
Next P

'Check for the existence of the homologizer files.
HomologyBaseName = Left(HP.DB, Len(HP.DB) - 4)
foo = FileLen(HP.Path & "\" & HomologyBaseName & ".headers")
foo = FileLen(HP.Path & "\" & HomologyBaseName & ".seqs")

'Initialize the homologizer.
If (InitPH(HP.Path & "\" & HomologyBaseName & ".seqs", NumSeqs, HP.Seed,
MaxProbeLen) <> 0) Then _
    Err.Raise pperrdll, , "Error in InitPH DLL Call."

'Load the probes into the homologizer index.
For P = 0 To NumSeqs - 1
    If (AddProbe(Seq(P), Len(Seq(P))) <> 0) Then _
        Err.Raise pperrdll, , "Error in AddProbe DLL Call."
Next P

'Open the specified DB.
DBFile = FreeFile
Open HP.Path & "\" & HomologyBaseName & ".headers" For Input As #DBFile

'Read the number of entries expected.  The progressbar for this application
'is posted with a maximum of 100, since we don't know the number of sequences
```

```
'in advance of opening the DB.
Line Input #DBFile, LineText
NumHeaders = CLng(LineText)

'Create an array of all accessions that should not be searched against.
NumNoCheck = 0
If IsGoodRS(RSBlast) Then
    RSBlast.MoveLast
    RSBlast.MoveFirst
    NumNoCheck = RSBlast.RecordCount
End If
ReDim NoCheck(0 To NumNoCheck)
N = 0
Do While Not RSBlast.EOF
    If Not RSBlast("Use") Then
        NoCheck(N) = RSBlast("Accession")
        N = N + 1
    End If
    RSBlast.MoveNext
Loop
NumNoCheck = N

'Homologize all the sequences in the DB.
S = 0
Do While Not EOF(DBFile)
    S = S + 1
    If (Fix(100 * S / NumHeaders) - Progress.StopAt > 0) Then
Progress.CheckProgress Fix(100 * S / NumHeaders)
    'Read the next DB sequence.
    SR = ReadFASTAHeader(DBFile)
    'Check the header against the bad list.
    DoHomology = True
    For N = 0 To NumNoCheck - 1
        If SR.Accession = NoCheck(N) Then DoHomology = False
    Next N
    'Do it.
    If DoHomology Then
        If (CalcOneHomology(SR.Length) <> 0) Then _
            Err.Raise pperrdll, , "Error in CalcOneHomology DLL Call."
    Else
        If (SkipSequence(SR.Length) <> 0) Then _
            Err.Raise pperrdll, , "Error in SkipSequence DLL Call."
    End If
Loop

'Retrieve the answers, and clean up.
If (GetHomology(Homology(0)) <> 0) Then _
    Err.Raise pperrdll, , "Error in GetHomology DLL Call."
If (TerminatePH() <> 0) Then _
    Err.Raise pperrdll, , "Error in TerminatePH DLL Call."

Exit Sub
E: Debug.Print "Error in CalcHomology"
Err.Raise Err.Number, , Err.Description
End Sub
```

```
Public Function BLASTPS&(PSName$, ByVal Sequence$, BP As cBLASTPars,
MatchAccession$(), _
    MatchHeader$(), MatchScores#(), MatchExpects#())
'-----------------------------------------------------------------------------
'Function
'   Run BLAST on the sequence of a probeset, saving headers and scores of
'   homologues.
'Arguments
'   PSName: The probeset name, used to name temporary files.
'   Sequence: The base sequence of the probeset.
'   BP: The BLAST search parameters.
'   MatchAccession: Matching accession numbers.
'   MatchHeader: The full matching header.
'   MatchScores: The match scores.
'   MatchExpects: The expected number of matches with such scores.
'Notes
'   This is a very fragile function, since it depends on the format of the
'   BLAST output file (checked only for version number), and relies on every
'   header containing an accession number (though one will be manufactured
'   if none is found).
'-----------------------------------------------------------------------------
On Error GoTo E

Dim BIFileName$, BOFileName$     'BLAST input and output files.
Dim BIFile&, BOFile&             'BLAST input and output file descriptor.
Dim BLASTBaseName$               'the root name for the BLAST files.
Dim cmd$                         'BLAST command string.
Dim ProcID                       'Process ID of running BLAST program.
Dim foo                          'Bit bucket.
Dim LineText$                    'One line of the BLAST output file.
Dim H&                           'Index the matches.
Dim GBWhere&                     'Location of the GenBank accession within the
match header.

'Check for the existence of all the BLAST files.  If files are absent, an error
will occur.
BLASTBaseName = Left(BP.DB, Len(BP.DB) - 4)
foo = FileLen(BP.Path & "\blastall.exe")
foo = FileLen(BP.Path & "\" & BLASTBaseName & ".nin")
foo = FileLen(BP.Path & "\" & BLASTBaseName & ".nhr")
foo = FileLen(BP.Path & "\" & BLASTBaseName & ".nsq")

'BLAST file names.
BIFileName = PSName & ".fsa"
BOFileName = PSName & ".out"

'Open the BLAST input file.
BIFile = FreeFile
Open BP.Path & "\" & BIFileName For Output As BIFile

'Write out the header.
Print #BIFile, "> PSName = " & PSName

'Write out the sequence.
Do While Len(Sequence) > 60
    Print #BIFile, Mid$(Sequence, 1, 60)
    Sequence = Right(Sequence, Len(Sequence) - 60)
Loop
```

```
Print #BIFile, Sequence

'Close input file.
Close BIFile

'Put together the command string to execute.
BOFileName = PSName & ".out"
cmd = "cmd /c cd " & BP.Path
cmd = cmd & " & blastall -p blastn -i " & BIFileName & " -d " & BLASTBaseName &
" -o " & BOFileName

'Run it and wait.
ProcID = Shell(cmd, vbHide)
foo = WaitOnProgram(ProcID)

'Open the output file.
BOFile = FreeFile
Open BP.Path & "\" & BOFileName For Input As #BOFile

'Check for correct version
Line Input #BOFile, LineText
If Left(LineText, 12) <> "BLASTN 2.0.2" Then Err.Raise ppErrBlastVersion

'Move down to MatchExpects.
On Err GoTo ErrNoSignificantMatch
Do
    Line Input #BOFile, LineText
Loop Until InStr(LineText, "Sequences producing significant alignments:") <> 0
On Error GoTo E

'Strip one more line
Line Input #BOFile, LineText

'Read match information, until a blank line.
Line Input #BOFile, LineText
H = 0
Do
    LineText = Right(LineText, Len(LineText) - 68)
    MatchScores(H) = CDbl(Left(LineText, 4))
    LineText = Right(LineText, Len(LineText) - 4)
    If IsNumeric(LineText) Then
        MatchExpects(H) = CDbl(LineText)
    Else
        MatchExpects(H) = 0#
    End If
    H = H + 1
    Line Input #BOFile, LineText
Loop Until Len(LineText) = 0

'We will return the number of matches.
BLASTPS = H - 1

'Read as many headers as we found matches.
For H = 0 To BLASTPS - 1

    'Strip lines until we see ">".
    Do
        Line Input #BOFile, LineText
    Loop Until Left(LineText, 1) = ">"

    'Push this line onto the header.
    MatchHeader(H) = LineText

    'Keep pushing on left-justified lines until we see "Length".
    Do
        Line Input #BOFile, LineText
        LineText = LTrim(LineText)
        If (Left(LineText, 6) = "Length") Then Exit Do
        MatchHeader(H) = MatchHeader(H) & " " & LineText
    Loop Until False

    'Find the accession number.
    GBWhere = InStr(MatchHeader(H), "/gb=")
    If (GBWhere = 0) Then
        MatchAccession(H) = "Z00000"
    Else
        GBWhere = GBWhere + 4
        Do
            MatchAccession(H) = MatchAccession(H) & Mid(MatchHeader(H), GBWhere,
1)
            GBWhere = GBWhere + 1
        Loop Until (Mid(MatchHeader(H), GBWhere, 1) = " ")
    End If
Next H

'Remove the BLAST files.
Close #BOFile
Kill BP.Path & "\" & BIFileName
Kill BP.Path & "\" & BOFileName

Exit Function

ErrNoSignificantMatch:
'No match was found, so we've skipped over the match information.
BLASTPS = 0
Close #BOFile
Kill BP.Path & "\" & BIFileName
Kill BP.Path & "\" & BOFileName
Exit Function

BLASTPS = 0
E: Debug.Print "Error in BLASTPS"
Err.Raise Err.Number, , Err.Description
End Function
```

```
Attribute VB_Name = "ColumnValidation"
Option Explicit

Public Sub ColumnsValidate(RSPS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Check the validity of columns.
'Arguments
'   RSPS: The parameter recordset for the ProbeSets being checked.
'Notes
```

```
'   1. A column is valid if the values of the parameters used
'      to calculate it are equal to the values of those parameters
'      in the current instance of the appropriate parameter class.
'   2. This calculation affects settings in the global state variable
'      PSColumns.  It should therefore only be called on the ProbeSets and
'      Probes used to build dbgProbes, i.e. datSelPS.
'   3. Filter columns are always valid, since they are initialized with True.
'-----------------------------------------------------------------------------

Dim CreatePSValid As Boolean
Dim LengthFilterValid As Boolean
Dim PosFilterValid As Boolean
Dim GCFilterValid As Boolean
Dim dGDValid As Boolean
Dim TMValid As Boolean
Dim dGHValid As Boolean
Dim dGMValid As Boolean
Dim ClampValid As Boolean
Dim RunValid As Boolean
Dim HomologyValid As Boolean
Dim Col As cColumn

'On startup, no columns exist, so bail.
If PSColumns.Count = 0 Then Exit Sub

'If the recordsets are empty, then nothing is valid.
If Not IsGoodRS(RSPS) Then
    For Each Col In PSColumns
        If Col.CanBeInValid Then Col.IsValid = False
    Next Col
    Exit Sub
End If

'Set everything valid.
CreatePSValid = True
LengthFilterValid = True
PosFilterValid = True
GCFilterValid = True
dGDValid = True
TMValid = True
dGHValid = True
dGMValid = True
RunValid = True
ClampValid = True
HomologyValid = True

'Try to invalidate.
RSPS.MoveFirst
Do While Not RSPS.EOF
    CreatePSValid = CreatePSValid And CreatePSPars.Validate(RSPS)
    LengthFilterValid = LengthFilterValid And LengthFilterPars.Validate(RSPS)
    PosFilterValid = PosFilterValid And PosFilterPars.Validate(RSPS)
    GCFilterValid = GCFilterValid And GCFilterPars.Validate(RSPS)
    dGDValid = dGDValid And dGDPars.Validate(RSPS) And
dGDFilterPars.Validate(RSPS)
    TMValid = TMValid And TMPars.Validate(RSPS) And TMFilterPars.Validate(RSPS)
    dGHValid = dGHValid And dGHPars.Validate(RSPS) And
dGHFilterPars.Validate(RSPS)
```

```
    dGMValid = dGMValid And dGMPars.Validate(RSPS) And
dGMFilterPars.Validate(RSPS)
    RunValid = RunValid And RunPars.Validate(RSPS) And
RunFilterPars.Validate(RSPS)
    ClampValid = ClampValid And ClampPars.Validate(RSPS) And
ClampFilterPars.Validate(RSPS)
    HomologyValid = HomologyValid And HomologyPars.Validate(RSPS) And
HomoFilterPars.Validate(RSPS)
    RSPS.MoveNext
Loop

'Update columns.
PSColumns("Sequence").IsValid = CreatePSValid
PSColumns("Length").IsValid = CreatePSValid And LengthFilterValid
PSColumns("Length Filter").IsValid = PSColumns("Length").IsValid
PSColumns("Position").IsValid = CreatePSValid And PosFilterValid
PSColumns("Pos Filter").IsValid = PSColumns("Position").IsValid
PSColumns("GC").IsValid = CreatePSValid And GCFilterValid
PSColumns("GC Filter").IsValid = PSColumns("GC").IsValid
PSColumns("Duplex dG").IsValid = dGDValid
PSColumns("dGD Filter").IsValid = PSColumns("Duplex dG").IsValid
PSColumns("TM").IsValid = TMValid
PSColumns("TM Filter").IsValid = PSColumns("TM").IsValid
PSColumns("dGH").IsValid = dGHValid
PSColumns("dGH Filter").IsValid = PSColumns("dGH").IsValid
PSColumns("dGM").IsValid = dGMValid
PSColumns("dGM Filter").IsValid = PSColumns("dGM").IsValid
PSColumns("Run Length").IsValid = RunValid
PSColumns("Run Filter").IsValid = PSColumns("Run Length").IsValid
PSColumns("Clamp").IsValid = ClampValid
PSColumns("Clamp Filter").IsValid = PSColumns("Clamp").IsValid
PSColumns("Homology").IsValid = HomologyValid
PSColumns("Homology Filter").IsValid = PSColumns("Homology").IsValid

'Set column heading backcolor according to validity.
For Each Col In PSColumns
    If Col.CanBeInValid Then
        If Col.IsValid Then
            frmMain.dbgProbes.Columns(Col.Name).HeadForeColor = vbBlack
        Else
            frmMain.dbgProbes.Columns(Col.Name).HeadForeColor = vbRed
        End If
    End If
Next Col

End Sub
Public Sub ColumnsExist(RSPS As Recordset, RSProbes As Recordset)
'---------------------------------------------------------------------------
'Function
'   Check the existence of columns.
'Arguments
'   RSPS: The probeset(s) parameter recordset.
'   PSProbes: The probeset probes.
'Notes
'   1. A column exists if
'      A. For each probeset, the appropriate existence parameter is set.
'         This indicates that some calculation has been performed to fill in
'         the column, at sometime in the past.  However, it is possible that
```

```
'         the selection criteria have changed since the calculation, bringing
'         into scope probes that have not had the calculation performed.  Thus,
'         the existence parameter is necessary, but not sufficient.
'      B. For each probe, the appropriate value is not NULL.  A lack of
'         null values is necessary and sufficient; however, it is potentially
'         time-consuming to check, so condition A is checked first, and
'         condition B is checked only if A passes.
'   2. This calculation affects settings in the global state variable
'      PSColumns.  It should therefore only be called on the ProbeSets and
'      Probes used to build dbgProbes, i.e. datSelPS and datSelPQuery.
'   3. Filter columns always exist, since they are initialized with True.
'---------------------------------------------------------------------------

Dim Exists As Boolean
Dim Col As cColumn

'On startup, no columns exist, so bail.
If PSColumns.Count = 0 Then Exit Sub

'If the recordsets are empty, then nothing exists, nothing is valid.
If Not (IsGoodRS(RSPS) And IsGoodRS(RSProbes)) Then
    For Each Col In PSColumns
        If Col.CanUnExist Then Col.Exists = False
    Next Col
    Exit Sub
End If

'Probe creation calculations.
Exists = True
RSPS.MoveFirst
Do While Not RSPS.EOF
    Exists = Exists And RSPS("CreatePS-Exists")
    RSPS.MoveNext
Loop
PSColumns("Accession").Exists = Exists
PSColumns("PSName").Exists = Exists
PSColumns("Sequence").Exists = Exists
PSColumns("Length").Exists = Exists
PSColumns("Position").Exists = Exists
PSColumns("GC").Exists = Exists

'TM Calculations.
Exists = True
RSPS.MoveFirst
Do While Not RSPS.EOF
    Exists = Exists And RSPS("TM-Exists")
    RSPS.MoveNext
Loop
If Exists = True Then
    RSProbes.FindFirst "TM = NULL"
    If Not RSProbes.NoMatch Then Exists = False
End If
PSColumns("TM").Exists = Exists

'dGH Calculations.
Exists = True
RSPS.MoveFirst
Do While Not RSPS.EOF
```

```
    Exists = Exists And RSPS("dGH-Exists")
    RSPS.MoveNext
Loop
If Exists = True Then
    RSProbes.FindFirst "dGH = NULL"
    If Not RSProbes.NoMatch Then Exists = False
End If
PSColumns("dGH").Exists = Exists

'dGM Calculations.
Exists = True
RSPS.MoveFirst
Do While Not RSPS.EOF
    Exists = Exists And RSPS("dGM-Exists")
    RSPS.MoveNext
Loop
If Exists = True Then
    RSProbes.FindFirst "dGM = NULL"
    If Not RSProbes.NoMatch Then Exists = False
End If
PSColumns("dGM").Exists = Exists

'Clamp Calculations.
Exists = True
RSPS.MoveFirst
Do While Not RSPS.EOF
    Exists = Exists And RSPS("Clamp-Exists")
    RSPS.MoveNext
Loop
If Exists = True Then
    RSProbes.FindFirst "[Clamp] = NULL"
    If Not RSProbes.NoMatch Then Exists = False
End If
PSColumns("Clamp").Exists = Exists

'Run Length Calculations.
Exists = True
RSPS.MoveFirst
Do While Not RSPS.EOF
    Exists = Exists And RSPS("Run-Exists")
    RSPS.MoveNext
Loop
If Exists = True Then
    RSProbes.FindFirst "[Run Length] = NULL"
    If Not RSProbes.NoMatch Then Exists = False
End If
PSColumns("Run Length").Exists = Exists

'Homology Calculations.
Exists = True
RSPS.MoveFirst
Do While Not RSPS.EOF
    Exists = Exists And RSPS("Homology-Exists")
    RSPS.MoveNext
Loop
If Exists = True Then
    RSProbes.FindFirst "[Homology] = NULL"
    If Not RSProbe.NoMatch Then Exists = False
```

```
End If
PSColumns("Homology").Exists = Exists

End Sub
```

```
Attribute VB_Name = "CreateProbeSet"
Option Explicit

Public Function CreatePSEqualL&(Seq$, CPSP As cCreatePSPars, Probes$(), _
    Positions&(), GC#())
'---------------------------------------------------------------------------
'Function:
'   Create a set of probes using the EqualLength method.
'Parameters:
'   Seq: The sequence from which the probes are created.
'   CPSP: An instance of the parameter class for this algorithm.
'   Probes: The array in which the probes will be returned.
'   Positions: The array in which the starting positions will be returned.
'   GC: The array in which GC content will be returned.
'Returns:
'   The number of probes created.
'Notes:
'   2. The calling routine is responsible for adequately dimensioning Probes,
'      Positions, and GC.
'---------------------------------------------------------------------------

Dim StartPos&     'the starting position of a probe
Dim ProbeNum&     'the current probe

StartPos = 1
ProbeNum = LBound(Probes)
Do While StartPos <= Len(Seq) - CPSP.Length + 1
    If ProbeNum - Progress.StopAt = 0 Then Progress.CheckProgress StartPos
    Probes(ProbeNum) = Mid(Seq, StartPos, CPSP.Length)
    Positions(ProbeNum) = StartPos
    GC(ProbeNum) = DNA_GCContent(Probes(ProbeNum))
    StartPos = StartPos + CPSP.Spacing
    ProbeNum = ProbeNum + 1
Loop
CreatePSEqualL = ProbeNum - 1
End Function

Public Function CreatePSEqualTM&(Seq$, CPSP As cCreatePSPars, Probes$(), _
    Positions&(), GC#())
'---------------------------------------------------------------------------
'Function:
'   Create a set of probes using the EqualTM method.
'Parameters:
'   Seq: The sequence from which the probes are created.
'   CPSP: An instance of the parameter class for this algorithm.
'   Probes: The array in which the probes will be returned.
'   Positions: The array in which the starting positions will be returned.
'   GC: The array in which GC content will be returned.
'Returns:
'   The number of probes created.
'Notes:
'   2. The calling routine is responsible for adequately dimensioning Probes,
'      Positions, and GC.
'------------------------------------------------------------------------------

Dim StartPos&      'the starting position of a probe
Dim ProbeNum&      'the current probe
Dim Length&        'current length of the probe being created
Dim TM#            'current TM of the probe being created

StartPos = 1
ProbeNum = LBound(Probes)
Do While StartPos <= Len(Seq)
    'Check progress
    If ProbeNum - Progress.StopAt = 0 Then Progress.CheckProgress StartPos
    'Create the probes, extending until melting point is above target.
    Length = 1
    Probes(ProbeNum) = Mid(Seq, StartPos, Length)
    Select Case CPSP.Duplex
    Case "DNA/DNA"
        TM = DNA_CalcTM(Probes(ProbeNum), CPSP.Conc)
    Case "DNA/RNA"
        TM = DR_CalcTM(Probes(ProbeNum), CPSP.Conc)
    End Select
    Length = Length + 1
    Do While TM < CPSP.Target
        If StartPos + Length > Len(Seq) Then
            ProbeNum = ProbeNum - 1
            Exit Do
        End If
        Probes(ProbeNum) = Mid(Seq, StartPos, Length)
        Select Case CPSP.Duplex
        Case "DNA/DNA"
            TM = DNA_CalcTM(Probes(ProbeNum), CPSP.Conc)
        Case "DNA/RNA"
            TM = DR_CalcTM(Probes(ProbeNum), CPSP.Conc)
        End Select
        Length = Length + 1
    Loop
    Positions(ProbeNum) = StartPos
    GC(ProbeNum) = DNA_GCContent(Probes(ProbeNum))
    ProbeNum = ProbeNum + 1
    StartPos = StartPos + CPSP.Spacing
Loop
CreatePSEqualTM = ProbeNum - 1
End Function
```

```
Attribute VB_Name = "DNA_Manipulations"
Option Explicit

Public Function DNA_GCContent#(ByVal Seq$)
'------------------------------------------------------------------------------
'Function
'   Compute the GC content of a sequence.
'------------------------------------------------------------------------------
Dim i&
Seq = LCase(Seq)
DNA_GCContent = 0
For i = 1 To Len(Seq)
```

```
    If (Mid(Seq, i, 1) = "g" Or Mid(Seq, i, 1) = "c") Then DNA_GCContent =
DNA_GCContent + 1
Next i
DNA_GCContent = DNA_GCContent / Len(Seq) * 100
End Function

Public Function DNA_SeqTrim$(Seq$, Bases&, WhichEnd$)
'------------------------------------------------------------------------------
'Function
'   Trim off one end or the other of the sequence.
'------------------------------------------------------------------------------
If Bases < Len(Seq) Then
    If WhichEnd = "5'" Then DNA_SeqTrim = Right(Seq, Len(Seq) - Bases)
    If WhichEnd = "3'" Then DNA_SeqTrim = Left(Seq, Len(Seq) - Bases)
End If
End Function

Public Function DNA_SeqKeep$(Seq$, Bases&, WhichEnd$)
'------------------------------------------------------------------------------
'Function
'   Remove all but the specified number of bases.
'------------------------------------------------------------------------------
If Bases <= Len(Seq) Then
    If WhichEnd = "5'" Then DNA_SeqKeep = Left(Seq, Bases)
    If WhichEnd = "3'" Then DNA_SeqKeep = Right(Seq, Bases)
End If
End Function

Public Function DNA_RevComp$(ByVal Seq$)
'------------------------------------------------------------------------------
'Function
'   Given 5'-3' strand, returns 5'-3' complement.
'Revision
'   28-Jul-97: From PkWs routine of the same name.
'------------------------------------------------------------------------------
Dim i&
Seq = LCase(Seq)
For i = Len(Seq) To 1 Step -1
    Select Case Mid$(Seq, i, 1)
    Case "a"
        DNA_RevComp = DNA_RevComp & "t"
    Case "c"
        DNA_RevComp = DNA_RevComp & "g"
    Case "g"
        DNA_RevComp = DNA_RevComp & "c"
    Case "t"
        DNA_RevComp = DNA_RevComp & "a"
    Case "u"
        DNA_RevComp = DNA_RevComp & "a"
    Case "n"
        DNA_RevComp = DNA_RevComp & "n"
    Case Else
        MsgBox "Unknown base found while calculating DNA_RevComp."
        DNA_RevComp = Seq
        Exit Function
    End Select
Next i
End Function
```

```
Public Function DNA_Comp$(ByVal Seq$)
'-----------------------------------------------------------------------------
'Function
'   Given 5'-3' strand, returns complement.  This might be useless!
'-----------------------------------------------------------------------------
Dim i&
Seq = LCase(Seq)
For i = 1 To Len(Seq)
    Select Case Mid$(Seq, i, 1)
    Case "a"
        DNA_Comp = DNA_Comp & "t"
    Case "c"
        DNA_Comp = DNA_Comp & "g"
    Case "g"
        DNA_Comp = DNA_Comp & "c"
    Case "t"
        DNA_Comp = DNA_Comp & "a"
    Case "u"
        DNA_Comp = DNA_Comp & "a"
    Case "n"
        DNA_Comp = DNA_Comp & "n"
    Case Else
        MsgBox "Unknown base found while calculating DNA_Comp."
        DNA_Comp = Seq
        Exit Function
    End Select
Next i
End Function

Public Function DNA_Rev$(ByVal Seq$)
'-----------------------------------------------------------------------------
'Function
'   Given 5'-3' strand, returns 3'-5' sequence.  This might be useless!
'-----------------------------------------------------------------------------
Dim i&
Seq = LCase(Seq)
For i = Len(Seq) To 1 Step -1
    Select Case Mid$(Seq, i, 1)
    Case "a"
        DNA_Rev = DNA_Rev & "a"
    Case "c"
        DNA_Rev = DNA_Rev & "c"
    Case "g"
        DNA_Rev = DNA_Rev & "g"
    Case "t"
        DNA_Rev = DNA_Rev & "t"
    Case "u"
        DNA_Rev = DNA_Rev & "u"
    Case "n"
        DNA_Rev = DNA_Rev & "n"
    Case Else
        MsgBox "Unknown base found while calculating DNA_Rev."
        DNA_Rev = Seq
        Exit Function
    End Select
Next i
End Function
```

```
Public Sub DNA_Str2Num(ByVal Seq$, ByRef NumSeq%())
'-----------------------------------------------------------------------------
'Function
'   Return a numeric array representing the DNA string.
'Arguments
'   Seq: The original sequence.
'   NumSeq: An integer array representing the sequence, w/ lower bound 0.
'Notes:
'   1. NumSeq must be correctly dimensioned by the calling routine.
'   since VB won't let me return an array from a function.
'-----------------------------------------------------------------------------
Dim i&
Seq = LCase(Seq)
For i = 0 To Len(Seq) - 1
    Select Case Mid(Seq, i + 1, 1)
    Case "a"
        NumSeq(i) = 0
    Case "c"
        NumSeq(i) = 1
    Case "g"
        NumSeq(i) = 2
    Case "t"
        NumSeq(i) = 3
    Case "u"
        NumSeq(i) = 3
    Case "n"
        NumSeq(i) = -1
    Case Else
        MsgBox "Unknown base found while calculating DNA_Str2Num."
    End Select
Next i
End Sub

Public Function DNA_Num2Str$(NumSeq%())
'-----------------------------------------------------------------------------
'Function:
'   Return the string representation of the DNA string
'Arguments:
'   NumSeq: An integer array representing the sequence.
'-----------------------------------------------------------------------------
Dim i&
For i = 0 To UBound(NumSeq)
    Select Case NumSeq(i)
    Case "0"
        DNA_Num2Str = DNA_Num2Str & "a"
    Case "1"
        DNA_Num2Str = DNA_Num2Str & "c"
    Case "2"
        DNA_Num2Str = DNA_Num2Str & "g"
    Case "3"
        DNA_Num2Str = DNA_Num2Str & "t"
    Case "-1"
        DNA_Num2Str = DNA_Num2Str & "n"
    Case Else
        MsgBox "Unknown numeric code found while calculating DNA_Num2Str."
    End Select
Next i
```

```
End Function
```

```
Attribute VB_Name = "Engine"
Option Explicit

Private Sub PSCalcBlast(RSPS As Recordset, RSBlast As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for BLAST calculation.
'Arguments
'   RSPS: The parameters recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                              'probe set name.
Dim Sequence$                            'sequence!
Dim MatchAccession$(MaxBLASTHeaders)
Dim MatchHeaders$(MaxBLASTHeaders)       'headers of matches
Dim MatchScores#(MaxBLASTHeaders)        'scores of matches
Dim MatchExpects#(MaxBLASTHeaders)       'expected value of matches
Dim H&                                   'loop index
Dim NumMatches&                          'number of matches found
Dim FindStr$

'Get the probe set name and sequence.
PSName = RSPS("PSName")
Sequence = RSPS("CreatePS-Sequence")
If Len(Sequence) = 0 Then GoTo UpdateOnly

'Run it.
Progress.ShowProgress "Calculating BLAST matches for ProbeSet " & PSName, 0, 1
NumMatches = BLASTPS(PSName, Sequence, BlastPars, MatchAccession, MatchHeaders,
MatchScores, MatchExpects)

'Update the results.
For H = 0 To NumMatches - 1
    With RSBlast
        'Check for this record already being present.
        .FindFirst "PSName = '" & PSName & "' AND Accession = '" &
MatchAccession(H) & "'"
        If .NoMatch Then
            .AddNew
        Else
            .Edit
        End If
        .Fields("PSName") = PSName
        .Fields("Accession") = MatchAccession(H)
        .Fields("Header") = MatchHeaders(H)
        .Fields("Score") = MatchScores(H)
        .Fields("Expect") = MatchExpects(H)
        If MatchScores(H) > BlastPars.SCutoff Or MatchExpects(H) <
BlastPars.ECutoff Then
            .Fields("Use") = False
        Else
            .Fields("Use") = True
        End If
        .Update
```

```
    End With
Next H

UpdateOnly:

'Store parameters in the database.
BlastPars.StoreDB RSPS

Exit Sub
E: Debug.Print "Error in PSCalcBlast"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub SeqCalcCreatePS(RSSeq As Recordset, RSProbes As Recordset, _
    RSPS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Create a ProbeSet
'Arguments
'   RSSeq: The sequence recordset (positioned to current sequence).
'   RSPS: The ProbeSets recordset (used to add a new ProbeSet).
'   RSProbes: The Probes recordset (used to add new probes).
'   RSPS: The Parameters recordset (used to add a new parameter record).
'Errors Raised
'   ppErrNoProbesCreated:  Raised if no probes are generated.
'Errors Handled
'   None.
'Notes
'   1. Under normal use (e.g. when called by the GUI), this routine uses the
'   values in CreatePSPars.
'   2. Only RSSeq must be a good recordset, since all the others are
'   used only to add records.
'-----------------------------------------------------------------------------

Dim Accession$      'current accession
Dim Seq$            'current sequence
Dim SeqLength&      'current sequence length
Dim NumPS&          'number of ProbeSets for this sequence
Dim PSName$         'new ProbeSet name
Dim Probes$()       'temporary store for created probes
Dim Positions&()    'temporary store for created positions
Dim GC#()           'temporary store for GC content
Dim NumProbes&      'number of probes created
Dim P&              'index variable for traversing Probes
Dim ProgressStr$    'text for progress bar

On Error GoTo E

'Check the recordset
If Not IsGoodRS(RSSeq) Then Exit Sub

'Get the sequence, accession, etc.
Accession = RSSeq("Accession")
Seq = RSSeq("Sequence")
SeqLength = RSSeq("Length")
NumPS = RSSeq("NumPS")

'New probeset name.
```

```
PSName = Accession & "_PS" & CStr(NumPS)

'Create temporary store for new probes.
ReDim Probes(SeqLength)
ReDim Positions(SeqLength)
ReDim GC(SeqLength)

'Open the progress bar.
Progress.ShowProgress "Creating ProbeSet " & PSName, 0, NumProbes

'Create the probes.
Select Case CreatePSPars.Method
Case "EqualL"
    NumProbes = CreatePSEqualL(Seq, CreatePSPars, Probes, Positions, GC)
Case "EqualTM"
    NumProbes = CreatePSEqualTM(Seq, CreatePSPars, Probes, Positions, GC)
End Select
If NumProbes = 0 Then Err.Raise ppErrNoProbesCreated
ReDim Preserve Probes(NumProbes)
ReDim Preserve Positions(NumProbes)
ReDim Preserve GC(NumProbes)

'Add a new ProbeSet.
RSSeq.Edit
RSSeq.Fields("NumPS") = NumPS + 1
RSSeq.Update
CreatePSPars.AddNewDB RSPS, Accession, PSName, SeqLength, Seq

'Create new records in the Probes table.
Progress.ShowProgress "Updating Probes in Database for Probeset " & PSName, 0,
NumProbes
For P = 0 To UBound(Probes) - 1
    If P - Progress.StopAt = 0 Then Progress.CheckProgress P
    RSProbes.AddNew
    RSProbes.Fields("Accession") = Accession
    RSProbes.Fields("PSName") = PSName
    RSProbes.Fields("Sequence") = Probes(P)
    RSProbes.Fields("Length") = Len(Probes(P))
    RSProbes.Fields("Position") = Positions(P)
    RSProbes.Fields("GC") = GC(P)
    RSProbes.Update
Next P

'Clean exit.
Exit Sub
E: Debug.Print "Error in SeqCalcCreatePS"
Err.Raise Err.Number, , Err.Description
End Sub

Public Sub SeqCalcEngine(Calc$)
'---------------------------------------------------------------------------------
'Function
'   SeqCalcEngine acts as the dispatcher for calculations
'   performed on Sequences.  Each of the known calculations
'   can be requested by specifying its name and subname.
'   SeqCalcEngine then retrieves necessary records from the
'   DB, then calls the specified calculation for each probe.
'Arguments
```

```
'   Calc: The calculation to perform.
'Notes
'   1. The parameters used for a particular calculation are
'      either made available in the optional parameter array
'      Pars, or retrieved from the class properties.
'   2. This routine loops over all selected Sequences.
'Errors
'   Errors are raised by not handled.  Calling routine is responsible
'---------------------------------------------------------------------------------

Dim RSSelSeqs As Recordset          'holds all selected Sequences
Dim RSP As Recordset                'holds all probes.
Dim RSAllProbes As Recordset        'holds all probes in one ProbeSet
Dim RSPS As Recordset               'holds parameters for the current ProbeSet
Dim PSName$                         'current ProbeSet name
Dim ErrNumber                       'saves number of errors raised by callee's
Dim ErrDescription                  'save description of errors raised by
callee's

'Error handling.
On Error GoTo E

'Check that some sequences are available.
Set RSSelSeqs = frmMain.datSelSeqs.Recordset
If Not IsGoodRS(RSSelSeqs) Then Exit Sub

'Map recordset pointers.
Set RSP = frmMain.datP.Recordset
Set RSPS = frmMain.datPS.Recordset

'Close the grid.
frmMain.dbgProbes.Close

'Process all selected sequences.
RSSelSeqs.MoveFirst
Do While Not RSSelSeqs.EOF

    'Begin a transaction block
    'XXXXX -- causes an error when entering this routine for the second time, no
idea why.
    'BeginTrans

    Select Case Calc

    Case "CreatePS"

        'Create the new probeset.
        SeqCalcCreatePS RSSelSeqs, RSP, RSPS

        'Get PS Name.
        PSName = RSPS("PSName")

        'Access all probes in this ProbeSet.
        With frmMain.datPOnePS
            .RecordSource = "SELECT * FROM Probes WHERE PSName = '" & PSName &
"'"
            .Refresh
            Set RSAllProbes = .Recordset
        End With

        'Compute length and position filters.
        If LengthFilterPars.Method <> "None" Then
            PSCalcLengthFilter RSAllProbes, RSPS
            PSColumns("Length Filter").IsFilter = LengthFilterPars.AppImmediate
        End If
        If PosFilterPars.Method <> "None" Then
            PSCalcPosFilter RSAllProbes, RSPS
            PSColumns("Pos Filter").IsFilter = PosFilterPars.AppImmediate
        End If
        If GCFilterPars.Method <> "None" Then
            PSCalcGCFilter RSAllProbes, RSPS
            PSColumns("GC Filter").IsFilter = GCFilterPars.AppImmediate
        End If
        PSColumns("PSName").IsVisible = True
        PSColumns("Sequence").IsVisible = True
        PSColumns("Length").IsVisible = True
        PSColumns("Position").IsVisible = True
        PSColumns("GC").IsVisible = True

    Case "3' Trim"
        SeqCalcTrim frmMain.datSelSeqs.Recordset, "3'"

    Case "5' Trim"
        SeqCalcTrim frmMain.datSelSeqs.Recordset, "5'"

    Case "3' Keep"
        SeqCalcKeep frmMain.datSelSeqs.Recordset, "3'"

    Case "5' Keep"
        SeqCalcKeep frmMain.datSelSeqs.Recordset, "5'"

    End Select

    'End the transaction.
    'CommitTrans

    'Move to next selected sequence.
    RSSelSeqs.MoveNext

Loop

'Update appearances.
frmMain.Form_ChangeSequence
frmMain.Form_ChangePSName

'Reopen the grid.
frmMain.dbgProbes.ReOpen
Progress.Hide

'Clean exit.
Exit Sub

'Handle errors.
E: Debug.Print "Error in SeqCalcEngine."
ErrNumber = Err.Number
ErrDescription = Err.Description
```

```
'Rollback
frmMain.Form_ChangeSequence
frmMain.Form_ChangePSName
frmMain.dbgProbes.ReOpen
Progress.Hide
frmMain.MousePointer = vbDefault
Err.Raise ErrNumber, , ErrDescription

End Sub


Private Sub SeqCalcTrim(RS As Recordset, WhichEnd$)
'--------------------------------------------------------------------------------
'Function
'   Trim bases from the 3' or 5' ends.
'--------------------------------------------------------------------------------
If Not IsGoodRS(RS) Then Exit Sub
RS.Edit
If WhichEnd = "5'" Then
    RS("Sequence") = DNA_SeqTrim(RS("sequence"), TrimPars.FivePTrim, "5'")
Else
    RS("Sequence") = DNA_SeqTrim(RS("sequence"), TrimPars.ThreePTrim, "3'")
End If
RS("Length") = Len(RS("Sequence"))
RS("Accession") = RS("Accession") & "*"
RS.Update
End Sub

Private Sub SeqCalcKeep(RS As Recordset, WhichEnd$)
'--------------------------------------------------------------------------------
'Function
'   Keep bases on the 3' or 5' ends.
'--------------------------------------------------------------------------------
If Not IsGoodRS(RS) Then Exit Sub
RS.Edit
If WhichEnd = "5'" Then
    RS("Sequence") = DNA_SeqKeep(RS("sequence"), TrimPars.FivePKeep, "5'")
Else
    RS("Sequence") = DNA_SeqKeep(RS("sequence"), TrimPars.ThreePKeep, "3'")
End If
RS("Length") = Len(RS("Sequence"))
RS("Accession") = RS("Accession") & "*"
RS.Update
End Sub

Private Sub SetField(RS As Recordset, Field$, Value)
'--------------------------------------------------------------------------------
'Function
'   Set all entries for one field in the recordset.
'--------------------------------------------------------------------------------
RS.MoveFirst
Do While Not RS.EOF
    RS.Edit
    RS.Fields(Field) = Value
    RS.Update
    RS.MoveNext
Loop
End Sub
```

```
Private Sub PutField(RS As Recordset, Field$, Value)
'---------------------------------------------------------------------------------
'Function
'   Put all entries for one field in the recordset.
'Notes
'   Since this operation can be timeconsuming, the progressbar is updated.
'   So a progressbar has to be posted first!
'---------------------------------------------------------------------------------
Dim P&
P = 0
RS.MoveFirst
Do While Not RS.EOF
    If P - Progress.StopAt = 0 Then Progress.CheckProgress P
    RS.Edit
    RS.Fields(Field) = Value(P)
    RS.Update
    P = P + 1
    RS.MoveNext
Loop
End Sub

Private Sub GetField(RS As Recordset, Field$, Values)
'---------------------------------------------------------------------------------
'Function
'   Get all entries for one field in the recordset.
'---------------------------------------------------------------------------------
Dim P&
P = 0
RS.MoveFirst
Do While Not RS.EOF
    Values(P) = RS.Fields(Field)
    RS.MoveNext
    P = P + 1
Loop
End Sub
Private Sub PSCalcdGH(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for dGH calculation.
'Arguments
'   RSProbes: The probes recordset (calculate dGH for all records).
'   RSPS: The parameters recordset (set parameters used in current record).
'---------------------------------------------------------------------------------
On Error GoTo E

Dim PSName$              'name of current probeset
Dim NumProbes&           'number of probes in the recordset
Dim Seq$()               'sequence column from database.
Dim dGH#()               'dGH column, to be put to database.

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating dGH for ProbeSet " & PSName, 0, NumProbes
```

```
'Create space for database fields.
ReDim Seq(0 To NumProbes - 1)
ReDim dGH(0 To NumProbes - 1)

'Get the sequences.
GetField RSProbes, "Sequence", Seq

'Calculate dGH.
Select Case dGHPars.DR
Case "DNA"
    DNA_CalcdGH Seq, dGHPars, dGH
Case "RNA"
    RNA_CalcdGH Seq, dGHPars, dGH
End Select

'Update the results.
Progress.ShowProgress "Updating dGH in Database for ProbeSet " & PSName, 0,
NumProbes
PutField RSProbes, "dGH", dGH

UpdateOnly:

'Store parameters in the database, and update column visibility.
dGHPars.StoreDB RSPS
PSColumns("dGH").Exists = True
PSColumns("dGH").IsVisible = True

'Calculate filter
PSCalcdGHFilter RSProbes, RSPS

Exit Sub
E: Debug.Print "Error in PSCalcdGH"
Err.Raise Err.Number, , Err.Description
End Sub


Private Sub PSCalcdGM(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for dGM calculation.
'Arguments
'   RSProbes: The probes recordset (calculate dGM for all records).
'   RSPS: The parameters recordset (set parameters used in current record).
'---------------------------------------------------------------------------------
On Error GoTo E

Dim PSName$              'name of current probeset
Dim NumProbes&           'number of probes in the recordset
Dim Seq$()               'sequence column from database.
Dim dGM#()               'dGM column, to be put to database.

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
```

```
Progress.ShowProgress "Calculating dGM for ProbeSet " & PSName, 0, NumProbes

'Create space for database fields.
ReDim Seq(0 To NumProbes - 1)
ReDim dGM(0 To NumProbes - 1)

'Get the sequences.
GetField RSProbes, "Sequence", Seq

'Calculate dGM.
Select Case dGMPars.DR
Case "DNA"
    DNA_CalcdGM Seq, dGMPars, dGM
Case "RNA"
    RNA_CalcdGM Seq, dGMPars, dGM
End Select

'Update the results.
Progress.ShowProgress "Updating dGM in Database for ProbeSet " & PSName, 0,
NumProbes
PutField RSProbes, "dGM", dGM

UpdateOnly:

'Store parameters in the database, and update column visibility.
dGMPars.StoreDB RSPS
PSColumns("dGM").Exists = True
PSColumns("dGM").IsVisible = True

'Calculate filter.
PSCalcdGMFilter RSProbes, RSPS

Exit Sub
E: Debug.Print "Error in PSCalcdGM."
Err.Raise Err.Number, , Err.Description
End Sub


Private Sub PSCalcRun(RSProbes As Recordset, RSPS As Recordset)
'------------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Run calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Run for all records).
'   RSPS: The parameters recordset (set parameters used in current record).
'------------------------------------------------------------------------------
On Error GoTo E

Dim PSName$             'name of current probeset
Dim NumProbes&          'number of probes in the recordset
Dim Pos&()              'position column from database.
Dim Run&()              'run column, to be put to database.

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly
```

```
'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Run for ProbeSet " & PSName, 0, NumProbes

'Create space for database fields.
ReDim Pos(0 To NumProbes - 1)
ReDim Run(0 To NumProbes - 1)

'Get the sequences.
GetField RSProbes, "Position", Pos

'Calculate Run.
CalcRun Pos, RunPars, Run

'Update the results.
Progress.ShowProgress "Updating Run in Database for ProbeSet " & PSName, 0,
NumProbes
PutField RSProbes, "Run Length", Run

UpdateOnly:

'Store parameters in the database, and update column visibility.
RunPars.StoreDB RSPS
PSColumns("Run Length").Exists = True
PSColumns("Run Length").IsVisible = True

'Calculate the run filter.
PSCalcRunFilter RSProbes, RSPS

Exit Sub
E: Debug.Print "Error in PSCalcRun"
Err.Raise Err.Number, , Err.Description
End Sub


Private Sub PSCalcHomology(RSProbes As Recordset, RSPS As Recordset, RSBlast As
Recordset)
'------------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Homology calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Homology for all records).
'   RSPS: The parameters recordset (set parameters used in current record).
'------------------------------------------------------------------------------
On Error GoTo E

Dim PSName$             'name of current probeset
Dim NumProbes&          'number of probes in the recordset
Dim Seq$()              'sequences of probes
Dim Homology&()         'run column, to be put to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
'Note that the progress bar in this case will show progress over sequences
'in the database.  Since we don't know the total, we'll set the maximum to
```

```
'100, and have the called routine do fraction calculations.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Homology for ProbeSet " & PSName, 0, 100

'Create space for database fields.
ReDim Seq(0 To NumProbes - 1)
ReDim Homology(0 To NumProbes - 1)

'Get the sequences.
GetField RSProbes, "Sequence", Seq

'Calculate Homology.
CalcHomology Seq, HomologyPars, Homology, RSBlast, PSName

'Update the results.
Progress.ShowProgress "Updating Homology in Database for ProbeSet " & PSName, 0,
NumProbes
PutField RSProbes, "Homology", Homology

UpdateOnly:

'Store parameters in the database, and update column visibility.
HomologyPars.StoreDB RSPS
PSColumns("Homology").Exists = True
PSColumns("Homology").IsVisible = True

'Calculate the homology filter.
PSCalcHomoFilter RSProbes, RSPS

Exit Sub
E: Debug.Print "Error in PSCalcHomology"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcTM(RSProbes As Recordset, RSPS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for TM calculation.
'Arguments
'   RSProbes: The probes recordset (calculate TM for all records).
'   RSPS: The parameters recordset (set parameters used in current record).
'-----------------------------------------------------------------------------
On Error GoTo E

Dim PSName$             'name of current probeset
Dim NumProbes&          'number of probes in the recordset
Dim Seq$()              'sequence column from database.
Dim TM#()               'TM column, to be put to database.

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating TM for ProbeSet " & PSName, 0, NumProbes

'Create space for database fields.
```

```
ReDim Seq(0 To NumProbes - 1)
ReDim TM(0 To NumProbes - 1)

'Get the sequences.
GetField RSProbes, "Sequence", Seq

'Calculate TM.
Select Case TMPars.Duplex
Case "DNA/DNA"
    DNA_CalcAllTM Seq, TMPars, TM
Case "DNA/RNA"
    DR_CalcAllTM Seq, TMPars, TM
End Select

'Update the results.
Progress.ShowProgress "Updating TM in Database for ProbeSet " & PSName, 0,
NumProbes
PutField RSProbes, "TM", TM

UpdateOnly:

'Store parameters in the database, and update column visibility.
TMPars.StoreDB RSPS
PSColumns("TM").Exists = True
PSColumns("TM").IsVisible = True

'Calculate the filter.
PSCalcTMFilter RSProbes, RSPS

Exit Sub
E: Debug.Print "Error in PSCalcTM"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcdGD(RSProbes As Recordset, RSPS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for dGD calculation.
'Arguments
'   RSProbes: The probes recordset (calculate dGD for all records).
'   RSPS: The parameters recordset (set parameters used in current record).
'-----------------------------------------------------------------------------
On Error GoTo E

Dim PSName$             'name of current probeset
Dim NumProbes&          'number of probes in the recordset
Dim Seq$()              'sequence column from database.
Dim dGD#()               'dGD column, to be put to database.

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating dG (Duplex) for ProbeSet " & PSName, 0,
NumProbes
```

```
'Create space for database fields.
ReDim Seq(0 To NumProbes - 1)
ReDim dGD(0 To NumProbes - 1)

'Get the sequences.
GetField RSProbes, "Sequence", Seq

'Calculate dGD.
Select Case dGDPars.Duplex
Case "DNA"
    DNA_CalcdGD Seq, dGDPars, dGD
Case "RNA"
    DR_CalcdGD Seq, dGDPars, dGD
End Select

'Update the results.
Progress.ShowProgress "Updating dG (Duplex) in Database for ProbeSet " & PSName,
0, NumProbes
PutField RSProbes, "Duplex dG", dGD

UpdateOnly:

'Store parameters in the database, and update column visibility.
dGDPars.StoreDB RSPS
PSColumns("Duplex dG").Exists = True
PSColumns("Duplex dG").IsVisible = True

'Calculate the filter.
PSCalcdGDFilter RSProbes, RSPS

Exit Sub
E: Debug.Print "Error in PSCalcdGD"
Err.Raise Err.Number, , Err.Description
End Sub


Private Sub PSCalcClamp(RSProbes As Recordset, RSPS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Clamp calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Clamp for all records).
'   RSPS: The parameters recordset (set parameters used in current record).
'-----------------------------------------------------------------------------
On Error GoTo E

Dim PSName$             'name of current probeset
Dim NumProbes&          'number of probes in the recordset
Dim Seq$()              'sequence column from database.
Dim Clamp#()              'Clamp column, to be put to database.

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Clamp for ProbeSet " & PSName, 0, NumProbes
```

```
'Create space for database fields.
ReDim Seq(0 To NumProbes - 1)
ReDim Clamp(0 To NumProbes - 1)

'Get the sequences.
GetField RSProbes, "Sequence", Seq

'Calculate Clamp.
Select Case ClampPars.Duplex
Case "DNA/DNA"
    DNA_CalcClamp Seq, ClampPars, Clamp
Case "DNA/RNA"
    'XXXXX
    DNA_CalcClamp Seq, ClampPars, Clamp
End Select

'Update the results.
Progress.ShowProgress "Updating Clamp in Database for ProbeSet " & PSName, 0,
NumProbes
PutField RSProbes, "Clamp", Clamp

UpdateOnly:

'Store parameters in the database, and update column visibility.
ClampPars.StoreDB RSPS
PSColumns("Clamp").Exists = True
PSColumns("Clamp").IsVisible = True

'Calculate the filter
PSCalcClampFilter RSProbes, RSPS

Exit Sub
E: Debug.Print "Error in PSCalcClamp"
Err.Raise Err.Number, , Err.Description
End Sub


Public Sub PSCalcEngine(Calc$)
'--------------------------------------------------------------------------------
'Function
'   PSCalcEngine acts as the dispatcher for calculations
'   performed on ProbeSets.  Each of the known calculations
'   can be requested by specifying its name.
'Arguments
'   Calc: The calculation to perform.
'Notes
'   1. All numeric calculations proceed by
'       - checking that the sequences column exists.
'       - checking that the current parameters set for this calculation
'         are the same as those previously used, and, if not, nulling
'         out all previous calculations.
'       - performing the calculation.
'   2. Filter calculations proceed as above, with the preliminary step
'   of checking the existence of the column to be filtered.  If it isn't
'   there, the calculation is done.
'   3. The Probes grid is closed before DB updates, then opened
'   when they are all over, to reduce screen thrashing.
'   4. Updates to the database are placed in transaction blocks, so
'   that they can be cleanly cancelled.  This should also improve
'   performance on network drives.
'   5. Calculation is always done one ProbeSet at a time, to reduce, whenever
'   possible, recalculation of values.  This could probably be further improved
'   by only recalculating NULL values (since any non-NULL is guarenteed good by
'   the above checks).
'Errors:
'   Errors are handled by rolling back pending DB transactions, cleaning up the
'   UI, then re-raising the same error for the caller to deal with.
'------------------------------------------------------------------------------

Dim RSSelPS As Recordset            'holds all selected ProbeSets
Dim RSAllProbes As Recordset        'holds all probes in one ProbeSet
Dim RSQProbes As Recordset          'holds probes in one ProbeSet that pass the
filters
Dim RSPS As Recordset               'holds parameters for the current ProbeSet
Dim RSBlast As Recordset            'holds BLAST homologies
Dim PSName$                         'current ProbeSet name
Dim ErrNumber                       'saves number of errors raised by callee's
Dim ErrDescription                  'save description of errors raised by
callee's

'Error handling.
On Error GoTo E

'Check that some ProbeSets are available, with sequence information.
Set RSSelPS = frmMain.datSelPS.Recordset
If Not IsGoodRS(RSSelPS) Then Exit Sub
If Not PSColumns("Sequence").Exists Then Exit Sub
Set RSBlast = frmEditBLAST.datBLAST.Recordset

'Close the grid.
frmMain.MousePointer = vbHourglass
frmMain.dbgProbes.Close

'Post the initial progress bar.
Progress.ShowProgress "Beginning Calculation: " & Calc, 0, 100

'Process all selected ProbeSets.
RSSelPS.MoveFirst
Do While Not RSSelPS.EOF

    'Get PS Name.
    PSName = RSSelPS("PSName")

    'Access all probes in this ProbeSet.
    With frmMain.datPOnePS
        .RecordSource = "SELECT * FROM Probes WHERE PSName = '" & PSName & "'"
        .Refresh
        Set RSAllProbes = .Recordset
    End With

    'Access probes in this ProbeSet that pass the filters.
    With frmMain.datPOnePSQuery
        .RecordSource = BuildPOnePSQuery(PSName)
        .Refresh
        Set RSQProbes = .Recordset
```

```
    End With

    'Position the parameter recordset for updates.
    With frmMain.datPS
        .Recordset.FindFirst "PSName = '" & PSName & "'"
        Set RSPS = .Recordset
    End With

    'Start a transaction block.
    BeginTrans

    'Choose the operation
    Select Case Calc

    Case "dGD"
        If dGDPars.Exists(RSPS) And Not dGDPars.Validate(RSPS) Then SetField
RSAllProbes, "dGD", Null
        If Not dGDFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGD
Filter", True
        PSCalcdGD RSQProbes, RSPS

    Case "dGD Filter"
        If Not PSColumns("dGD").Exists Then
            If dGDPars.Exists(RSPS) And Not dGDPars.Validate(RSPS) Then SetField
RSAllProbes, "dGD", Null
            If Not dGDFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGD
Filter", True
            PSCalcdGD RSQProbes, RSPS
        Else
            If Not dGDFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGD
Filter", True
            PSCalcdGDFilter RSQProbes, RSPS
        End If

    Case "TM"
        If TMPars.Exists(RSPS) And Not TMPars.Validate(RSPS) Then SetField
RSAllProbes, "TM", Null
        If Not TMFilterPars.Validate(RSPS) Then SetField RSAllProbes, "TM
Filter", True
        PSCalcTM RSQProbes, RSPS

    Case "TM Filter"
        If Not PSColumns("TM").Exists Then
            If TMPars.Exists(RSPS) And Not TMPars.Validate(RSPS) Then SetField
RSAllProbes, "TM", Null
            If Not TMFilterPars.Validate(RSPS) Then SetField RSAllProbes, "TM
Filter", True
            PSCalcTM RSQProbes, RSPS
        Else
            If Not TMFilterPars.Validate(RSPS) Then SetField RSAllProbes, "TM
Filter", True
            PSCalcTMFilter RSQProbes, RSPS
        End If

    Case "dGH"
        If dGHPars.Exists(RSPS) And Not dGHPars.Validate(RSPS) Then SetField
RSAllProbes, "dGH", Null
```

```
        If Not dGHFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGH
Filter", True
        PSCalcdGH RSQProbes, RSPS

    Case "dGH Filter"
        If Not PSColumns("dGH").Exists Then
            If dGHPars.Exists(RSPS) And Not dGHPars.Validate(RSPS) Then SetField
RSAllProbes, "dGH", True
            If Not dGHFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGH
Filter", True
            PSCalcdGH RSQProbes, RSPS
        Else
            If Not dGHFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGH
Filter", True
            PSCalcdGHFilter RSQProbes, RSPS
        End If

    Case "dGM"
        If dGMPars.Exists(RSPS) And Not dGMPars.Validate(RSPS) Then SetField
RSAllProbes, "dGM", Null
        If Not dGMFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGM
Filter", True
        PSCalcdGM RSQProbes, RSPS

    Case "dGM Filter"
        If Not PSColumns("dGM").Exists Then
            If dGMPars.Exists(RSPS) And Not dGMPars.Validate(RSPS) Then SetField
RSAllProbes, "dGM", True
            If Not dGMFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGM
Filter", True
            PSCalcdGM RSQProbes, RSPS
        Else
            If Not dGMFilterPars.Validate(RSPS) Then SetField RSAllProbes, "dGM
Filter", True
            PSCalcdGMFilter RSQProbes, RSPS
        End If

    Case "Run"
        If RunPars.Exists(RSPS) And Not RunPars.Validate(RSPS) Then SetField
RSAllProbes, "Run Length", Null
        If Not RunFilterPars.Validate(RSPS) Then SetField RSAllProbes, "Run
Filter", True
        PSCalcRun RSQProbes, RSPS

    Case "Run Filter"
        If Not PSColumns("Run Length").Exists Then
            If RunPars.Exists(RSPS) And Not RunPars.Validate(RSPS) Then SetField
RSAllProbes, "Run", True
            If Not RunFilterPars.Validate(RSPS) Then SetField RSAllProbes, "Run
Filter", True
            PSCalcRun RSQProbes, RSPS
        Else
            If Not RunFilterPars.Validate(RSPS) Then SetField RSAllProbes, "Run
Filter", True
            PSCalcRunFilter RSQProbes, RSPS
        End If

    Case "Clamp"
```

```
        If ClampPars.Exists(RSPS) And Not ClampPars.Validate(RSPS) Then SetField
RSAllProbes, "Clamp", Null
        If Not ClampFilterPars.Validate(RSPS) Then SetField RSAllProbes, "Clamp
Filter", True
        PSCalcClamp RSQProbes, RSPS

    Case "Clamp Filter"
        If Not PSColumns("Clamp").Exists Then
            If ClampPars.Exists(RSPS) And Not ClampPars.Validate(RSPS) Then
SetField RSAllProbes, "Clamp", Null
            If Not ClampFilterPars.Validate(RSPS) Then SetField RSAllProbes,
"Clamp Filter", True
            PSCalcClamp RSQProbes, RSPS
        Else
            If Not ClampFilterPars.Validate(RSPS) Then SetField RSAllProbes,
"Clamp Filter", True
            PSCalcClampFilter RSQProbes, RSPS
        End If

    Case "Length Filter"
        If Not PSColumns("Length").Exists Then Exit Sub
        If Not LengthFilterPars.Validate(RSPS) Then SetField RSAllProbes,
"Length Filter", True
        PSCalcLengthFilter RSQProbes, RSPS

    Case "Pos Filter"
        If Not PSColumns("Position").Exists Then Exit Sub
        If Not PosFilterPars.Validate(RSPS) Then SetField RSAllProbes, "Pos
Filter", True
        PSCalcPosFilter RSQProbes, RSPS

    Case "GC Filter"
        If Not PSColumns("GC").Exists Then Exit Sub
        If Not GCFilterPars.Validate(RSPS) Then SetField RSAllProbes, "GC
Filter", True
        PSCalcGCFilter RSQProbes, RSPS

    Case "BLAST"
        PSCalcBlast RSPS, RSBlast

    Case "Homology"
        If HomologyPars.Exists(RSPS) And Not HomologyPars.Validate(RSPS) Then
SetField RSAllProbes, "Homology", Null
        If Not HomoFilterPars.Validate(RSPS) Then SetField RSAllProbes,
"Homology Filter", True
        PSCalcHomology RSQProbes, RSPS, RSBlast

    Case "Homology Filter"
        If Not PSColumns("Homology").Exists Then
            If HomologyPars.Exists(RSPS) And Not HomologyPars.Validate(RSPS)
Then SetField RSAllProbes, "Homology", Null
            If Not HomoFilterPars.Validate(RSPS) Then SetField RSAllProbes,
"Homology Filter", True
            PSCalcHomology RSQProbes, RSPS, RSBlast
        Else
            If Not HomoFilterPars.Validate(RSPS) Then SetField RSAllProbes,
"Homology Filter", True
            PSCalcHomoFilter RSQProbes, RSPS
        End If

    End Select

    'End the transaction.
    CommitTrans

    'Move to next selected probeset.
    RSSelPS.MoveNext

Loop

'Reopen the grid.
Progress.Hide
frmMain.dbgProbes.ReOpen
DoEvents

'Update appearances.
frmMain.Form_ChangePSName

'Clean exit.
frmMain.MousePointer = vbDefault
Exit Sub

'Handle errors.
E: Debug.Print "Error in PSCalcEngine."
ErrNumber = Err.Number
ErrDescription = Err.Description
Rollback
frmMain.Form_ChangePSName
frmMain.dbgProbes.ReOpen
Progress.Hide
frmMain.MousePointer = vbDefault
Err.Raise ErrNumber, , ErrDescription

End Sub


Private Sub PSCalcTMFilter(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for TM filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate TM Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                   'name of current probeset
Dim NumProbes&                'number of probes in the recordset
Dim TM#()                     'TM column from database
Dim TMFilter() As Boolean     'TM Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly
```

```
'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating TM Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim TM(0 To NumProbes - 1)
ReDim TMFilter(0 To NumProbes - 1)

'Get the TMs.
GetField RSProbes, "TM", TM

'Calculate TM Filter.
CalcTMFilter TM, TMFilterPars, TMFilter

'Update the results.
Progress.ShowProgress "Updating TM Filter in Database for ProbeSet " & PSName,
0, NumProbes
PutField RSProbes, "TM Filter", TMFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
TMFilterPars.StoreDB RSPS
PSColumns("TM Filter").IsFilter = TMFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcTMFilter"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcdGDFilter(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for dGD filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate dGD Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                   'name of current probeset
Dim NumProbes&                'number of probes in the recordset
Dim dGD#()                    'dGD column from database
Dim dGDFilter() As Boolean    'dGD Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating dG (Duplex) Filter for ProbeSet " & PSName,
0, NumProbes

'Create space for database fields.
ReDim dGD(0 To NumProbes - 1)
ReDim dGDFilter(0 To NumProbes - 1)
```

```
'Get the dGDs.
GetField RSProbes, "Duplex dG", dGD

'Calculate dGD Filter.
CalcdGDFilter dGD, dGDFilterPars, dGDFilter

'Update the results.
Progress.ShowProgress "Updating dG (Duplex) Filter in Database for ProbeSet " &
PSName, 0, NumProbes
PutField RSProbes, "dGD Filter", dGDFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
dGDFilterPars.StoreDB RSPS
PSColumns("dGD Filter").IsFilter = dGDFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcdGDFilter"
Err.Raise Err.Number, , Err.Description
End Sub


Private Sub PSCalcClampFilter(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Clamp filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Clamp Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                  'name of current probeset
Dim NumProbes&               'number of probes in the recordset
Dim Clamp#()                   'Clamp column from database
Dim ClampFilter() As Boolean   'Clamp Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Clamp Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim Clamp(0 To NumProbes - 1)
ReDim ClampFilter(0 To NumProbes - 1)

'Get the Clamp values.
GetField RSProbes, "Clamp", Clamp

'Calculate Clamp Filter.
CalcClampFilter Clamp, ClampFilterPars, ClampFilter
```

```
'Update the results.
Progress.ShowProgress "Updating Clamp Filter in Database for ProbeSet " &
PSName, 0, NumProbes
PutField RSProbes, "Clamp Filter", ClampFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
ClampFilterPars.StoreDB RSPS
PSColumns("Clamp Filter").IsFilter = ClampFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcClampFilter"
Err.Raise Err.Number, , Err.Description
End Sub
Private Sub PSCalcdGHFilter(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for dGH filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate dGH Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                  'name of current probeset
Dim NumProbes&               'number of probes in the recordset
Dim dGH#()                   'dGH column from database
Dim dGHFilter() As Boolean   'dGH Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating dGH Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim dGH(0 To NumProbes - 1)
ReDim dGHFilter(0 To NumProbes - 1)

'Get the dGHs.
GetField RSProbes, "dGH", dGH

'Calculate dGH Filter.
CalcdGHFilter dGH, dGHFilterPars, dGHFilter

'Update the results.
Progress.ShowProgress "Updating dGH Filter in Database for ProbeSet " & PSName,
0, NumProbes
PutField RSProbes, "dGH Filter", dGHFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
dGHFilterPars.StoreDB RSPS
```

```
PSColumns("dGH Filter").IsFilter = dGHFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcdGHFilter"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcGCFilter(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for GC filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate GC Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                   'name of current probeset
Dim NumProbes&                'number of probes in the recordset
Dim GC#()                     'GC column from database
Dim GCFilter() As Boolean     'GC Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating GC Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim GC(0 To NumProbes - 1)
ReDim GCFilter(0 To NumProbes - 1)

'Get the GCs.
GetField RSProbes, "GC", GC

'Calculate GC Filter.
CalcGCFilter GC, GCFilterPars, GCFilter

'Update the results.
Progress.ShowProgress "Updating GC Filter in Database for ProbeSet " & PSName,
0, NumProbes
PutField RSProbes, "GC Filter", GCFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
GCFilterPars.StoreDB RSPS
PSColumns("GC Filter").IsFilter = GCFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcGCFilter"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcdGMFilter(RSProbes As Recordset, RSPS As Recordset)
```

```
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for dGM filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate dGM Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                   'name of current probeset
Dim NumProbes&                'number of probes in the recordset
Dim dGM#()                    'dGM column from database
Dim dGMFilter() As Boolean    'dGM Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating dGM Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim dGM(0 To NumProbes - 1)
ReDim dGMFilter(0 To NumProbes - 1)

'Get the dGMs.
GetField RSProbes, "dGM", dGM

'Calculate dGM Filter.
CalcdGMFilter dGM, dGMFilterPars, dGMFilter

'Update the results.
Progress.ShowProgress "Updating dGM Filter in Database for ProbeSet " & PSName,
0, NumProbes
PutField RSProbes, "dGM Filter", dGMFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
dGMFilterPars.StoreDB RSPS
PSColumns("dGM Filter").IsFilter = dGMFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcdGMFilter"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcRunFilter(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Run filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Run Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E
```

```
Dim PSName$                  'name of current probeset
Dim NumProbes&               'number of probes in the recordset
Dim Run&()                   'Run column from database
Dim RunFilter() As Boolean   'Run Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Run Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim Run(0 To NumProbes - 1)
ReDim RunFilter(0 To NumProbes - 1)

'Get the Runs.
GetField RSProbes, "Run Length", Run

'Calculate Run Filter.
CalcRunFilter Run, RunFilterPars, RunFilter

'Update the results.
Progress.ShowProgress "Updating Run Filter in Database for ProbeSet " & PSName,
0, NumProbes
PutField RSProbes, "Run Filter", RunFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
RunFilterPars.StoreDB RSPS
PSColumns("Run Filter").IsFilter = RunFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcRunFilter"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcHomoFilter(RSProbes As Recordset, RSPS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Homology filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Homology Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'-----------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                  'name of current probeset
Dim NumProbes&               'number of probes in the recordset
Dim Homology&()                  'Homology column from database
Dim HomoFilter() As Boolean   'Homology Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
```

```
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Homology Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim Homology(0 To NumProbes - 1)
ReDim HomoFilter(0 To NumProbes - 1)

'Get the Homologys.
GetField RSProbes, "Homology", Homology

'Calculate Homology Filter.
CalcHomoFilter Homology, HomoFilterPars, HomoFilter

'Update the results.
Progress.ShowProgress "Updating Homology Filter in Database for ProbeSet " &
PSName, 0, NumProbes
PutField RSProbes, "Homology Filter", HomoFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
HomoFilterPars.StoreDB RSPS
PSColumns("Homology Filter").IsFilter = HomoFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcHomoFilter"
Err.Raise Err.Number, , Err.Description
End Sub


Private Sub PSCalcLengthFilter(RSProbes As Recordset, RSPS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Length filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Length Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'-----------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                      'name of current probeset
Dim NumProbes&                   'number of probes in the recordset
Dim Length&()                    'Length column from database
Dim LengthFilter() As Boolean    'Length Filter column to database

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Length Filter for ProbeSet " & PSName, 0,
NumProbes
```

```
'Create space for database fields.
ReDim Length(0 To NumProbes - 1)
ReDim LengthFilter(0 To NumProbes - 1)

'Get the Lengths.
GetField RSProbes, "Length", Length

'Calculate Length Filter.
CalcLengthFilter Length, LengthFilterPars, LengthFilter

'Update the results.
Progress.ShowProgress "Updating Length Filter in Database for ProbeSet " &
PSName, 0, NumProbes
PutField RSProbes, "Length Filter", LengthFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
LengthFilterPars.StoreDB RSPS
PSColumns("Length Filter").IsFilter = LengthFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcLengthFilter"
Err.Raise Err.Number, , Err.Description
End Sub

Private Sub PSCalcPosFilter(RSProbes As Recordset, RSPS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Perform all database gets/puts, etc, for Pos filter calculation.
'Arguments
'   RSProbes: The probes recordset (calculate Pos Filter for all records).
'   RSPS: The parameter recordset (set parameters used in current record).
'---------------------------------------------------------------------------
On Error GoTo E

Dim PSName$                   'name of current probeset
Dim NumProbes&                'number of probes in the recordset
Dim Pos&()                    'Pos column from database
Dim PosFilter() As Boolean    'Pos Filter column to database
Dim SeqLength&                'Length of the originating sequence.

'Determine the number of probes.
NumProbes = NumRecords(RSProbes)
If NumProbes = 0 Then GoTo UpdateOnly

'Start it up.
PSName = RSProbes("PSName")
Progress.ShowProgress "Calculating Pos Filter for ProbeSet " & PSName, 0,
NumProbes

'Create space for database fields.
ReDim Pos(0 To NumProbes - 1)
ReDim PosFilter(0 To NumProbes - 1)

'Get the Positions.
GetField RSProbes, "Position", Pos
```

```
'Get the sequence length.
SeqLength = RSPS("CreatePS-SeqLength")

'Calculate Pos Filter.
CalcPosFilter Pos, SeqLength, PosFilterPars, PosFilter

'Update the results.
Progress.ShowProgress "Updating Pos Filter in Database for ProbeSet " & PSName,
0, NumProbes
PutField RSProbes, "Pos Filter", PosFilter

UpdateOnly:

'Store parameters in the database, and update column visibility.
PosFilterPars.StoreDB RSPS
PSColumns("Pos Filter").IsFilter = PosFilterPars.AppImmediate

Exit Sub
E: Debug.Print "Error in PSCalcPosFilter"
Err.Raise Err.Number, , Err.Description
End Sub
```

```
Attribute VB_Name = "Filters"
Option Explicit

Public Sub CalcTMFilter(TM#(), TMFPars As cTMFilterPars, Filter() As Boolean)
'---------------------------------------------------------------------------
'Function
'   Decide which probes pass the TM filter.
'Arguments
'   TM: An array of TMs
'   TMFPars: An instance of the parameter class cTMFilterPars.
'   Filter: The return array of filter output values.
'---------------------------------------------------------------------------

'The TM filter can be implemented directly by the generic filter.
CalcGenericFilter TM, TMFPars, Filter
End Sub


Public Sub CalcdGDFilter(dGD#(), dGDFPars As cdGDFilterPars, Filter() As
Boolean)
'---------------------------------------------------------------------------
'Function
'   Decide which probes pass the dGD filter.
'Arguments
'   dGD: An array of dGDs
'   dGDFPars: An instance of the parameter class cdGDFilterPars.
'   Filter: The return array of filter output values.
'---------------------------------------------------------------------------

'The dGD filter can be implemented directly by the generic filter.
CalcGenericFilter dGD, dGDFPars, Filter
End Sub
```

```
Public Sub CalcGenericFilter(Values, FilterPars As Object, Filter() As Boolean)
'--------------------------------------------------------------------------
'Function
'   Decide which values pass the filter.
'Arguments
'   Values: An array of values to be filtered against.
'   FilterPars: The filter parameters
'   Filter: The returned filter settings.
'Notes
'   This routine is used to implement the common filter methods.  FilterPars is
'   declared as an object rather than a specific class, and thus needs to have
'   only the members accessed on the execution path.
'   FilterPars must have data member Method.
'   If method is None, no other members are required.
'   If method is Min, then member Min is required.
'   If method is Max, then member Max is required.
'   If method is InRange, then members Max and Min are required.
'   If method is Distance, then members Target and Distance are required.
'   If method is NBest, then members Target and NBest are required.
'   If method is Percent, then members Target and Percent are required.
'History
'   31-Jul-97: PW
'--------------------------------------------------------------------------

Dim Distance#()              'absolute difference between values and target
Dim Target#                  'target value
Dim MinValue#, MaxValue#     'min and max of value
Dim Index&()                 'index array to track sort
Dim NumValues&               'number of values we are working with
Dim T&                       'index
Dim N&                       'number to set.

'Determine the number of probes we are calculating filter for.
NumValues = UBound(Values) + 1

'If method is "None", set all to true.
If FilterPars.Method = "None" Then
    For T = 0 To NumValues - 1
        Filter(T) = True
    Next T
    Exit Sub
End If

'If method is "Min", set all >= min to true.
If FilterPars.Method = "Min" Then
    For T = 0 To NumValues - 1
        Filter(T) = (Values(T) >= Val(FilterPars.Min))
    Next T
    Exit Sub
End If

'If method is "Max", set all <= max to true.
If FilterPars.Method = "Max" Then
    For T = 0 To NumValues - 1
        Filter(T) = (Values(T) <= Val(FilterPars.Max))
    Next T
    Exit Sub
End If
```

```
'If method is "InRange", set all >= min and <= max to true.
If FilterPars.Method = "InRange" Then
    For T = 0 To NumValues - 1
        Filter(T) = ((Values(T) >= Val(FilterPars.Min)) And (Values(T) <=
Val(FilterPars.Max)))
    Next T
    Exit Sub
End If

'For other methods, we have to do more work!

'Size the arrays.
ReDim Distance(NumValues - 1)
ReDim Index(NumValues - 1)

'Find the minimum and maximum values.
For T = 0 To NumValues - 1
    If MinValue > Values(T) Then MinValue = Values(T)
    If MaxValue < Values(T) Then MaxValue = Values(T)
Next T

'Choose the target value.
If (FilterPars.Method = "Distance") Or _
   (FilterPars.Method = "NBest") Or _
   (FilterPars.Method = "Percent") _
   Then Target = FilterPars.Target
If (FilterPars.Method = "NLowest") Or _
   (FilterPars.Method = "PercentLowest") _
   Then Target = MinValue
If (FilterPars.Method = "NHighest") Or _
   (FilterPars.Method = "PercentHighest") _
   Then Target = MaxValue

'Compute distances of Values from target; set up index and filter.
For T = 0 To NumValues - 1
    Distance(T) = Abs(Values(T) - Target)
    Index(T) = T
    Filter(T) = False
Next T

'If method is "Distance", set all elements within distance as true.
If FilterPars.Method = "Distance" Then
    For T = 0 To NumValues - 1
        Filter(T) = Distance(T) <= FilterPars.Distance
    Next T
    Exit Sub
End If

'Choose the number to set
If (FilterPars.Method = "NBest") Or _
   (FilterPars.Method = "NLowest") Or _
   (FilterPars.Method = "NHighest") _
   Then N = FilterPars.NBest
If (FilterPars.Method = "Percent") Or _
   (FilterPars.Method = "PercentLwest") Or _
   (FilterPars.Method = "PercentHighest") _
   Then N = NumValues * FilterPars.Percent / 100 - 1
```

```
If N >= NumValues Then N = NumValues - 1

'Sort on distance from target.
QuickSort Distance, Index, 0, NumValues - 1

'Set the best.
For T = 0 To N - 1
    Filter(Index(T)) = True
Next T

End Sub


Public Sub CalcRunFilter(Run&(), RunFPars As CRunFilterPars, Filter() As
Boolean)
'-----------------------------------------------------------------------------
'Function
'   Decide which probes pass the Run filter.
'Arguments
'   Run: An array of Runs
'   RunFPars: An instance of the parameter class cRunFilterPars.
'   Filter: The return array of filter output values.
'-----------------------------------------------------------------------------
Dim OldMin&

'The special method "All" can be handled by using the method Min, with min=1.
If RunFPars.Method = "All" Then
    OldMin = RunFPars.Min
    RunFPars.Min = 1
    RunFPars.Method = "Min"
    CalcGenericFilter Run, RunFPars, Filter
    RunFPars.Method = "All"
    RunFPars.Min = OldMin
    Exit Sub
End If

'Otherwise, filters are generic.
CalcGenericFilter Run, RunFPars, Filter

End Sub


Public Sub CalcdGMFilter(dGM#(), dGMFPars As cdGMFilterPars, Filter() As
Boolean)
'-----------------------------------------------------------------------------
'Function
'   Decide which probes pass the dGM filter.
'Arguments
'   dGM: An array of dGMs
'   dGMFPars: An instance of the parameter class cdGMFilterPars.
'   Filter: The return array of filter output values.
'-----------------------------------------------------------------------------

CalcGenericFilter dGM, dGMFPars, Filter
End Sub
```

```
Public Sub CalcClampFilter(Clamp#(), ClampFPars As cClampFilterPars, Filter() As
Boolean)
'-----------------------------------------------------------------------------
'Function
'   Decide which probes pass the Clamp Filter.
'Arguments
'   Clamp: An array of Clamps
'   ClampFPars: An instance of the parameter class cClampFilterPars.
'   Filter: The return array of filter output values.
'Notes
'History
'   31-Jul-97: PW
'-----------------------------------------------------------------------------

CalcGenericFilter Clamp, ClampFPars, Filter
End Sub

Public Sub CalcGCFilter(GC#(), GCFPars As cGCFilterPars, Filter() As Boolean)
'-----------------------------------------------------------------------------
'Function
'   Decide which probes pass the GC filter.
'Arguments
'   GC: An array of GCs
'   GCFPars: An instance of the parameter class cGCFilterPars.
'   Filter: The return array of filter output values.
'-----------------------------------------------------------------------------

CalcGenericFilter GC, GCFPars, Filter
End Sub

Public Sub CalcdGHFilter(dGH#(), dGHFPars As cdGHFilterPars, Filter() As
Boolean)
'-----------------------------------------------------------------------------
'Function
'   Decide which probes pass the dGH filter.
'Arguments
'   dGH: An array of dGHs
'   dGHFPars: An instance of the parameter class cdGHFilterPars.
'   Filter: The return array of filter output values.
'-----------------------------------------------------------------------------

CalcGenericFilter dGH, dGHFPars, Filter
End Sub


Public Sub CalcHomoFilter(Homology&(), HFPars As cHomoFilterPars, Filter() As
Boolean)
'-----------------------------------------------------------------------------
'Function
'   Decide which probes pass the homology filter.
'Arguments
'   Homology: An array of homologiess
'   HFPars: An instance of the parameter class cHomoFilterPars.
'   Filter: The return array of filter output values.
'-----------------------------------------------------------------------------

CalcGenericFilter Homology, HFPars, Filter
```

```
End Sub

Public Sub CalcPosFilter(Pos&(), SeqLength&, PFPars As cPosFilterPars, Filter()
As Boolean)
'---------------------------------------------------------------------------
'Function
'   Decide which probes pass the position filter.
'Arguments
'   Pos: An array of positions.
'   SeqLength: The length of the sequence from which the probeset was derived.
'   PFPars: An instance of the parameter class cPosFilterPars.
'   Filter: The return array of filter output values.
'---------------------------------------------------------------------------

'The target for this filter should be the specified end.
If PFPars.WhichEnd = "5-prime" Then
    PFPars.Target = 1
Else
    PFPars.Target = SeqLength
End If
CalcGenericFilter Pos, PFPars, Filter
End Sub

Public Sub CalcLengthFilter(Length&(), LFPars As cLengthFilterPars, Filter() As
Boolean)
'---------------------------------------------------------------------------
'Function
'   Decide which probes pass the length filter.
'Arguments
'   Length: An array of lengths. .
'   LFPars: An instance of the parameter class cLengthFilterPars.
'   Filter: The return array of filter output values.
'---------------------------------------------------------------------------
CalcGenericFilter Length, LFPars, Filter
End Sub
```

```
Attribute VB_Name = "GenBank"
Option Explicit

Private Const GBTerminator$ = "//"      'record terminator in Genbank files
Private Const MaxGBFileLength& = 100000 'maximum allowed GenBank file length

Public Sub EntrezLoadDB(Accession$, RS As Recordset, KeepFile As Boolean)
'---------------------------------------------------------------------------
'Function
'   Grab a sequence using the Entrez CGI.
'Arguments
'   Accession: The accession of the required sequence.
'   RS: The recordset into which this sequence should be loaded.
'   KeepFile: Controls whether the downloaded file is retained or removed.
'---------------------------------------------------------------------------
On Error GoTo E

Dim GBFileName$                 'Genbank sequence file for data.
Dim EntrezData$, EntrezData2$   'GB Entries.
Dim i&
```

```
'Grab the data.
EntrezData = frmEntrezGrab.Entrez.GetGenBankData(Accession, "")

'Create CRLFs out of LFs.
For i = 1 To Len(EntrezData)
    If (Mid$(EntrezData, i, 1) = vbLf) Then EntrezData2 = EntrezData2 + vbCr
    EntrezData2 = EntrezData2 + Mid$(EntrezData, i, 1)
Next i

'Write the aquired data out to a file.
GBFileName = frmMain.cdgSeqs.InitDir & "\" & Accession & ".gb"
Open GBFileName For Output As #1
Print #1, EntrezData2
Close #1

'Load the database.
SeqLoadDB ReadGBRecord(GBFileName), RS

'Remove the file.
If Not KeepFile Then Kill GBFileName

Exit Sub
E: Debug.Print "Error in EntrezLoadDB"
Err.Raise Err.Number, , Err.Description
End Sub

Public Sub SeqLoadDB(SeqRC As SeqRecord, RS As Recordset)
'---------------------------------------------------------------------------
'Function
'   Create a new record in the Sequence table, populate from SeqRC
'Arguments
'   SeqRC: A sequence structure, as read from file, etc.
'   RS: The recordset into which the sequence should be loaded.
'Errors:
'   Attempting to create a new sequence record whose Accession matches a
'   previously inserted sequence causes a DB error.
'---------------------------------------------------------------------------
On Error GoTo E

BeginTrans
    AllowdbgClick = False
    RS.AddNew
    RS.Fields("Header") = SeqRC.Header
    RS.Fields("Accession") = SeqRC.Accession
    RS.Fields("Locus") = SeqRC.Locus
    RS.Fields("Length") = SeqRC.Length
    RS.Fields("Sequence") = SeqRC.Sequence
    RS.Fields("Selected") = True
    RS.Update
    RS.Bookmark = RS.LastModified
CommitTrans
DoEvents
AllowdbgClick = True
Exit Sub
E: Debug.Print "Error in SeqLoadDB"
Rollback
DoEvents
```

```
AllowdbgClick = True
Err.Raise Err.Number, , Err.Description
End Sub

Public Function ReadGBRecord(FileName$) As SeqRecord
'-------------------------------------------------------------------------------
'Function
'   Read a GenBank record from a file into a SeqRecord.
'Arguments
'   FileName: The filename of the GB record.
'Returns
'   The GenBank record as a SeqRecord.
'Errors:
'   Returned to caller.
'   1. Check file is less than MaxGBFileLength
'   2. Check that all required fields are present.
'-------------------------------------------------------------------------------
On Error GoTo E

Dim GBFile&             'GenBank file
Dim FileLength&         'file length (chars)
Dim LineLength&         'length of current line (chars)

Dim GBText$             'entire record
Dim GBItem$             'one line
Dim KeyField$           'item keyfield
Dim foo$                'bit bucket

Dim GB As SeqRecord     'the returned record

'open the file
GBFile = FreeFile
Open FileName For Binary As #GBFile

'read in the file as one long string
FileLength = FileLen(FileName)
If FileLength > MaxGBFileLength Then Err.Raise ppErrGBFileLength
GBText = Input(FileLength, #GBFile)

'check for required fields
If InStr(GBText, "LOCUS") = 0 Then Err.Raise ppErrGBFileFormat, , "No LOCUS
field found in GB file."
If InStr(GBText, "ACCESSION") = 0 Then Err.Raise ppErrGBFileFormat, , "No
ACCESSION field found in GB file."
If InStr(GBText, GBTerminator) = 0 Then Err.Raise ppErrGBFileFormat, , "No
terminator (//) found in GB file."

'process items
Do While True

    'extract an item
    LineLength = InStr(GBText, vbCrLf) - 1
    GBItem = Left(GBText, LineLength)
    GB.Header = GB.Header & GBItem & vbCrLf
    GBText = Right(GBText, FileLength - LineLength - 2)
    FileLength = FileLength - LineLength - 2

    If Left(GBItem, 2) = GBTerminator Then
```

```
        GoTo breakwhile1
    End If

    'extract keyfield, and process item
    KeyField = StrField(GBItem, LineLength, 10)
    'if keyfield is not numeric -> header
    If IsNumeric(KeyField) = False Then
        'if keyfield occupies first column -> keyword
        If Left(KeyField, 1) <> " " Then
            'process keywords
            Select Case KeyField
            Case "LOCUS"
                foo = StrField(GBItem, LineLength, 2)
                GB.Locus = StrField(GBItem, LineLength, 10)
                GB.Length = StrField(GBItem, LineLength, 7)
            Case "ACCESSION"
                foo = StrField(GBItem, LineLength, 2)
                GB.Accession = StrField(GBItem, LineLength, 6)
            End Select
        'if keyfield has only two blanks -> subkeyword
        ElseIf Left(KeyField, 3) <> "   " Then
        'if keyfield has only 4 blanks -> feature code
        ElseIf Left(KeyField, 5) <> "     " Then
        'it must be a continuation line
        Else
        End If
    Else
        'sequence has started
        Do While LineLength > 10
            GB.Sequence = GB.Sequence & Left(GBItem, 10)
            GBItem = Right(GBItem, LineLength - 11)
            LineLength = LineLength - 11
        Loop
        If LineLength > 0 Then
            GB.Sequence = GB.Sequence & GBItem
        End If
    End If
Loop
breakwhile1:
Close GBFile

'Strip any whitespace.
GB.Sequence = PackSequence(GB.Sequence)

'Check sequence length.
If GB.Length <> Len(GB.Sequence) Then
    MsgBox "Sequence length differs from header specification."
    GB.Length = Len(GB.Sequence)
    GB.Accession = GB.Accession & "*"
End If

ReadGBRecord = GB
Exit Function
E: Debug.Print "Error in ReadGBRecord"
Err.Raise Err.Number, , Err.Description
End Function
```

```
Public Function ReadFASTARecord(FASTAFile&) As SeqRecord
'-------------------------------------------------------------------------------
'Function
'   Read a FASTA record from a file into a SeqRecord.
'Arguments
'   FASTA: The file descriptor for the FASTA file.
'Returns
'   The FASTA record as a SeqRecord.
'Errors:
'   Returned to caller.
'Notes:
'   The FASTA file must already be opened for input, and should be positioned
'   at the start of a FASTA header.  It will read an entire sequence, leaving
'   the file positioned on the subsequent header.
'-------------------------------------------------------------------------------
On Error GoTo E

Dim LineText$           'one line of the file
Dim SR As SeqRecord     'the returned record

'Read the header line, check it is legal.
Line Input #FASTAFile, LineText
If Left(LineText, 1) <> ">" Then Err.Raise ppErrFASTAFormat, , "Header line does
not begin w/ >."

'Extract the length.
LineText = Right(LineText, Len(LineText) - InStr(LineText, "len=") - 3)
If InStr(LineText, "/") <> 0 Then LineText = Left(LineText, InStr(LineText, "/")
- 1)
SR.Length = CLng(LineText)

'Read sequence lines till we are done.
Do While (Len(SR.Sequence) <> SR.Length)
    Line Input #FASTAFile, LineText
    SR.Sequence = SR.Sequence & LineText
Loop

ReadFASTARecord = SR

Exit Function
E: Debug.Print "Error in ReadFASTARecord"
Err.Raise Err.Number, , Err.Description
End Function

Public Function ReadFASTAHeader(FASTAFile&) As SeqRecord
'-------------------------------------------------------------------------------
'Function
'   Read a FASTA header from a file into a SeqRecord.
'Arguments
'   FASTA: The file descriptor for the FASTA header file.
'Returns
'   The FASTA record as a SeqRecord.
'Errors:
'   Returned to caller.
'Notes:
'   The FASTA header file must already be opened for input, and should be
positioned
'   at the start of a FASTA header.
```

```
'-------------------------------------------------------------------------------
On Error GoTo E

Dim LineText$           'one line of the file
Dim LenText$            'line processed to find length
Dim AccText$            'line processed to find accession
Dim SR As SeqRecord     'the returned record

'Read the header line, check it is legal.
Line Input #FASTAFile, LineText
If Left(LineText, 1) <> ">" Then Err.Raise ppErrFASTAFormat, , "Header line does
not begin w/ >."

'Extract the length.
LenText = Right(LineText, Len(LineText) - InStr(LineText, "len=") - 3)
If InStr(LenText, "/") <> 0 Then LenText = Left(LenText, InStr(LenText, "/") -
1)
SR.Length = CLng(LenText)

'Extract the accession.
AccText = Right(LineText, Len(LineText) - InStr(LineText, "gb=") - 2)
If InStr(AccText, "/") <> 0 Then AccText = Left(AccText, InStr(AccText, "/") -
1)
SR.Accession = LTrim(RTrim(AccText))

ReadFASTAHeader = SR

Exit Function
E: Debug.Print "Error in ReadFASTARecord"
Err.Raise Err.Number, , Err.Description
End Function


Public Function GBCookSeq(ByVal RawSeq$) As String
'Convert raw sequence (no spaces, etc) into more
'palatable form, with spaces every 10 nucleotides
'and newlines every 60 nucleotides.

Dim Bases&          'count the bases
Dim LineLength&     'remaining linelength

LineLength = Len(RawSeq)
Bases = 1
Do While LineLength > 60
    GBCookSeq = GBCookSeq & _
        Format(Format(Bases, "#####"), "@@@@@") & " "
    Bases = Bases + 60
    Dim i&
    For i = 1 To 6
        GBCookSeq = GBCookSeq & StrField(RawSeq, LineLength, 10) & " "
    Next i
    GBCookSeq = GBCookSeq & vbCrLf
Loop
If Len(RawSeq) <> 0 Then
    GBCookSeq = GBCookSeq & _
        Format(Format(Bases, "#####"), "@@@@@") & " "
    Do While Len(RawSeq) > 10
        GBCookSeq = GBCookSeq & StrField(RawSeq, LineLength, 10) & " "
    Loop
    GBCookSeq = GBCookSeq & RawSeq
End If
End Function

Public Function RTFGBCookSeq(yVal RawSeq$) As String
'Convert raw sequence (no spaces, etc) into more
'palatable form, with spaces every 10 nucleotides
'and newlines every 60 nucleotides.

Dim Bases&          'count the bases
Dim LineLength&     'remaining linelength

LineLength = Len(RawSeq)
Bases = 1
Do While LineLength > 60
    RTFGBCookSeq = RTFGBCookSeq & _
        Format(Format(Bases, "#####"), "@@@@@") & " "
    Bases = Bases + 60
    Dim i&
    For i = 1 To 6
        RTFGBCookSeq = RTFGBCookSeq & StrField(RawSeq, LineLength, 10) & " "
    Next i
    RTFGBCookSeq = RTFGBCookSeq & "\par"
Loop
If Len(RawSeq) <> 0 Then
    RTFGBCookSeq = RTFGBCookSeq & _
        Format(Format(Bases, "#####"), "@@@@@") & " "
    Do While Len(RawSeq) > 10
        RTFGBCookSeq = RTFGBCookSeq & StrField(RawSeq, LineLength, 10) & " "
    Loop
    RTFGBCookSeq = RTFGBCookSeq & RawSeq
End If
End Function
```

```
Attribute VB_Name = "SQL"
Option Explicit

Public Function BuildPOnePSQuery(PSName)
'---------------------------------------------------------------
'Function
'   Build an SQL Query to select Probes from the Probes table
'   that pass all filters, and come from the named ProbeSet.
'Arguments
'   Accession: The accession to search on.
'   PSName: The ProbeSet name to search on.
'Notes
'   2. All fields corresponding to a column (that is, all fields
'      named in the Columns table) are returned, to match the layout
'      of the Probes grid.  The visibility of various columns is
'      controlled by the column Visible properties.
'   3. The currently set filters are used as a further selection
'      on the probes returned.
'---------------------------------------------------------------
Dim SelectC$, FromC$, FilterC$, WhereC$, OrderByC$
Dim C&, F&
```

```
'Set up the select clause -- return all columns.
SelectC = "SELECT * "
'For C = 1 To PSColumns.Count
'    If C <> 1 Then SelectC = SelectC & ", "
'    SelectC = SelectC & "[" & PSColumns(C).Name & "] "
'Next C

'Set up the the from clause -- always the Probes table.
FromC = "FROM Probes "

'Set up the where clause
For F = 1 To PSColumns.Count
    If (PSColumns(F).IsFilter = True) Then
        If FilterC = "" Then
            FilterC = "([" & PSColumns(F).Name & "] = True)"
        Else
            FilterC = FilterC & " AND ([" & PSColumns(F).Name & "] = True)"
        End If
    End If
Next F
WhereC = "WHERE ( (PSName = '" & PSName & "')"
If PSColumns("User Filter").IsFilter Then
    If FilterC <> "" Then
        WhereC = WhereC & " AND ( ([User Filter] = 2) OR ( ([User Filter] = 1)
AND " & FilterC & ") ) ) "
    Else
        WhereC = WhereC & " AND ([User Filter] <> 0)) ) "
    End If
Else
    If FilterC <> "" Then
        WhereC = WhereC & " AND ( " & FilterC & " ) ) "
    Else
        WhereC = WhereC & " ) "
    End If
End If

'Put it all together.
BuildPOnePSQuery = SelectC & FromC & WhereC & ";"


End Function

Public Function BuildProbesQuery()
'---------------------------------------------------------------
'Function
'   Build an SQL Query to select Probes from the Probes table.
'Arguments
'   Accession: The accession to search on.
'   PSName: The ProbeSet name to search on.
'Notes
'   1. Accession or PSName = <none> is allowed, and results
'      in an empty recordset.
'   2. All fields corresponding to a column (that is, all fields
'      named in the Columns table) are returned, to match the layout
'      of the Probes grid.  The visibility of various columns is
'      controlled by the column Visible properties.
'   3. The currently set filters are used as a further selection
'      on the probes returned.
```

```
'----------------------------------------------------------------
Dim SelectC$, FromC$, FilterC$, WhereC$, OrderByC$
Dim C&, F&
Dim Col As cColumn

'Set up the select clause -- return all columns.
SelectC = "SELECT * "
'For C = 1 To PSColumns.Count
'    If C <> 1 Then SelectC = SelectC & ", "
'    SelectC = SelectC & "Probes.[" & PSColumns(C).Name & "] "
'Next C

'Set up the the from clause -- from the SelectProbes query.
FromC = "FROM SelectProbes "

'Set up the where clause -- filter the rows.
'For F = 1 To PSColumns.count
For Each Col In PSColumns
    If (Col.IsFilter = True) Then
        If FilterC = "" Then
            FilterC = "([" & Col.Name & "] = True)"
        Else
            FilterC = FilterC & " AND ([" & Col.Name & "] = True)"
        End If
    End If
Next Col
WhereC = "WHERE ( (Selected = True)"
If PSColumns("User Filter").IsFilter Then
    If FilterC <> "" Then
        WhereC = WhereC & " AND ( ([User Filter] = 2) OR ( ([User Filter] = 1)
AND " & FilterC & ") ) ) "
    Else
        WhereC = WhereC & " AND ([User Filter] <> 0]) ) "
    End If
Else
    If FilterC <> "" Then
        WhereC = WhereC & " AND ( " & FilterC & " ) ) "
    Else
        WhereC = WhereC & " ) "
    End If
End If

'Set up the orderby clause.
'Default to sorting by position, then check columns.
OrderByC = "Position "
For Each Col In PSColumns
    If (Col.IsSort = "Ascending") Then OrderByC = "[" & Col.Name & "] ASC "
    If (Col.IsSort = "Descending") Then OrderByC = "[" & Col.Name & "] DESC "
Next Col
OrderByC = "ORDER BY Probes.Accession, Probes.PSName, " & OrderByC

'Put it all together.
BuildProbesQuery = SelectC & FromC & WhereC & OrderByC & ";"

End Function
```

```
Attribute VB_Name = "Stats"
```

```
Option Explicit
Public Type StatRecord
    Count As Long
    Min As Double
    Ten As Double
    Median As Double
    Ninety As Double
    Max As Double
    Range As Double
    Average As Double
    Std As Double
End Type


Public Function Statistics(Vector#()) As StatRecord
'------------------------------------------------------------------------
'Function
'   Calculate all statistics of a vector.
'------------------------------------------------------------------------
Dim L&, U&
Dim Min#, Max#, Total#, Total2#
Dim Index&()
Dim i&

'Get array bounds.
L = LBound(Vector)
U = UBound(Vector)

'Set the count.
Statistics.Count = UBound(Vector) - LBound(Vector) + 1
ReDim Index(L To U)

'Take a pass through to get min, max, total, total^2, index.
Min = Vector(L)
Max = Vector(L)
For i = L To U
    Index(i) = i
    Total = Total + Vector(i)
    Total2 = Total2 + Vector(i) * Vector(i)
    If Vector(i) > Max Then Max = Vector(i)
    If Vector(i) < Min Then Min = Vector(i)
Next i

'Set min, max, range, average, std.
Statistics.Min = Min
Statistics.Max = Max
Statistics.Range = Max - Min
Statistics.Average = Total / Statistics.Count
Statistics.Std = Sqr((Total2 / Statistics.Count) - Statistics.Average *
Statistics.Average)

'Sort.
QuickSort Vector, Index, L, U

'Percentiles.
Statistics.Ten = Vector(L + (Statistics.Count - 1) * 0.1)
Statistics.Median = Vector(L + (Statistics.Count - 1) * 0.5)
Statistics.Ninety = Vector(L + (Statistics.Count - 1) * 0.9)
```

```
End Function
```

```
Attribute VB_Name = "Thermo"
Option Explicit

'There is no good reason to have two sets of thermodynamic parameters,
'other than history...

'Parameters used by TM calculations.
Public Const RGas# = 1.987                  'Gas constant, cal/mol/deg K.
Private DNA_DuplexH#(0 To 3, 0 To 3)        'DNA/DNA nearest neighbor
enthalpies.
Private DNA_DuplexS#(0 To 3, 0 To 3)        'DNA/RNA nearest neighbor entropies.
Private DNA_InitGCS#, DNA_InitATS#          'DNA/DNA initiation entropies.
Private DNA_SelfS#                          'DNA/DNA self-symmetry entropies.
Private DNA_EndTAH#                         'DNA/DNA TA end enthalpy.
Private DR_DuplexH#(0 To 3, 0 To 3)         'DNA/RNA nearest neighbor
enthalpies.
Private DR_DuplexS#(0 To 3, 0 To 3)         'DNA/RNA nearest neighbor entropies.
Private DR_InitH#, DR_InitS#                'DNA/RNA initiation
enthaply/entropy.

'Parameters used by hairpin calculations.
'Zuker provides H and G @ 37.  So on load, calculate S, then recalculate G as
needed.
Private Const NumZukerLoops& = 30             'Number of special case loops.
Private ZLoopLengths&(0 To NumZukerLoops - 1)  'The special case loop lengths.
Private Const DNA_ZMiscLoop# = 1.079
Private Const RNA_ZMiscLoop# = 1.079          'Constants for large-loop
entropies.
Private Const ZMinLoop& = 4                   'Minumum allowed loop.
Private Const NumTetraLoops& = 8              'Number of special tetraloops.
Private ZTetraLoops&(0 To NumTetraLoops - 1)  'Indices of tetraloops.

'DNA free energy.
Private DNAHP_StackG#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_TStackG#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_DangleG#(0 To 1, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_LoopG#(0 To 2, 0 To NumZukerLoops - 1)
Private DNAHP_TLoopG#(0 To NumTetraLoops - 1)

'DNA enthalpy.
Private DNAHP_StackH#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_TStackH#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_DangleH#(0 To 1, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_LoopH#(0 To 2, 0 To NumZukerLoops - 1)
Private DNAHP_TLoopH#(0 To NumTetraLoops - 1)

'DNA entropy.
Private DNAHP_StackS#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_TStackS#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_DangleS#(0 To 1, 0 To 3, 0 To 3, 0 To 3)
Private DNAHP_LoopS#(0 To 2, 0 To NumZukerLoops - 1)
Private DNAHP_TLoopS#(0 To NumTetraLoops - 1)

'RNA free energy.
```

```
Private RNAHP_StackG#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_TStackG#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_DangleG#(0 To 1, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_LoopG#(0 To 2, 0 To NumZukerLoops - 1)
Private RNAHP_TLoopG#(0 To NumTetraLoops - 1)

'RNA enthalpy.
Private RNAHP_StackH#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_TStackH#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_DangleH#(0 To 1, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_LoopH#(0 To 2, 0 To NumZukerLoops - 1)
Private RNAHP_TLoopH#(0 To NumTetraLoops - 1)

'RNA entropy.
Private RNAHP_StackS#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_TStackS#(0 To 3, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_DangleS#(0 To 1, 0 To 3, 0 To 3, 0 To 3)
Private RNAHP_LoopS#(0 To 2, 0 To NumZukerLoops - 1)
Private RNAHP_TLoopS#(0 To NumTetraLoops - 1)

Private Sub CalcSFromGH(Dims&, G#(), H#(), ByVal T#, S#())
'-----------------------------------------------------------------------------
'Function
'   Calculate S given G, H, and T, for all elements of array S.
'Arguments
'   Dims: The number of dimensions of G, H, and S.
'   G: The given free energies.
'   H: The given enthalpies.
'   T: The temperature at which free energy was calculated.
'   S: The returned entropies.
'Notes
'   1. The reason for this functions existence is that Zuker provides
'   G at 37, and H.  S is needed to change temperatures.
'-----------------------------------------------------------------------------
Dim I&, J&, K&, L&
T = T + 273.15
Select Case Dims
Case 1
    For I = LBound(G) To UBound(G)
        S(I) = (G(I) - H(I)) / T
    Next I
Case 2
    For I = LBound(G, 1) To UBound(G, 1)
        For J = LBound(G, 2) To UBound(G, 2)
            S(I, J) = (G(I, J) - H(I, J)) / T
        Next J
    Next I
Case 4
    For I = LBound(G, 1) To UBound(G, 1)
        For J = LBound(G, 2) To UBound(G, 2)
            For K = LBound(G, 3) To UBound(G, 3)
                For L = LBound(G, 4) To UBound(G, 4)
                    S(I, J, K, L) = (G(I, J, K, L) - H(I, J, K, L)) / T
                Next L
            Next K
        Next J
    Next I
End Select
End Sub

Private Sub CalcGFromHS(Dims&, H#(), S#(), ByVal T#, G#())
'-----------------------------------------------------------------------------
'Function
'   Calculate G given H, S, and T, for all elements of array G.
'Arguments
'   Dims: The number of dimensions of G, H, and S.
'   H: The given enthalpies.
'   S: The given entropies.
'   T: The temperature at which free energy is required.
'   S: The returned free energies.
'-----------------------------------------------------------------------------
Dim I&, J&, K&, L&
T = T + 273.15
Select Case Dims
Case 1
    For I = LBound(G) To UBound(G)
        G(I) = 0.1 * CDbl(CInt(10# * (H(I) + S(I) * T)))
    Next I
Case 2
    For I = LBound(G, 1) To UBound(G, 1)
        For J = LBound(G, 2) To UBound(G, 2)
            G(I, J) = 0.1 * CDbl(CInt(10# * (H(I, J) + S(I, J) * T)))
        Next J
    Next I
Case 4
    For I = LBound(G, 1) To UBound(G, 1)
        For J = LBound(G, 2) To UBound(G, 2)
            For K = LBound(G, 3) To UBound(G, 3)
                For L = LBound(G, 4) To UBound(G, 4)
                    G(I, J, K, L) = 0.1 * CDbl(CInt(10# * (H(I, J, K, L) + S(I,
J, K, L) * T)))
                Next L
            Next K
        Next J
    Next I
End Select
End Sub

Private Function DNA_BestHairpin#(Seq%())
'-----------------------------------------------------------------------------
'Function
'   Compute the most stable hairpin for the given sequence.
'Arguments
'   Seq: The numerical representation of the sequence.
'-----------------------------------------------------------------------------
Dim ThisHairpin#
Dim NumBases&
Dim FP&, TP&

DNA_BestHairpin = 1000#
NumBases = UBound(Seq) - LBound(Seq) + 1

For FP = 0 To NumBases - ZMinLoop - 2
    For TP = FP + ZMinLoop + 1 To NumBases - 1
        ThisHairpin = DNA_Hairpin(Seq, FP, TP)
        If ThisHairpin < DNA_BestHairpin Then DNA_BestHairpin = ThisHairpin
```

```
    Next TP
Next FP

End Function

Private Function RNA_BestHairpin#(Seq%())
'-----------------------------------------------------------------------------
'Function
'   Compute the most stable hairpin for the given sequence.
'Arguments
'   Seq: The numerical representation of the sequence.
'-----------------------------------------------------------------------------
Dim ThisHairpin#
Dim NumBases&
Dim FP&, TP&

RNA_BestHairpin = 1000#
NumBases = UBound(Seq) - LBound(Seq) + 1

For FP = 0 To NumBases - ZMinLoop - 2
    For TP = FP + ZMinLoop + 1 To NumBases - 1
        ThisHairpin = RNA_Hairpin(Seq, FP, TP)
        If ThisHairpin < RNA_BestHairpin Then RNA_BestHairpin = ThisHairpin
    Next TP
Next FP

End Function


Private Function RNA_Hairpin#(Seq%(), ByVal FP&, ByVal TP&)
'-----------------------------------------------------------------------------
'Function
'   Calculate the free energy of a RNA hairpin
'Arguments
'   Seq: The sequence, in numeric representation.
'   FP: The index of the 5'-end base that closes the loop.
'   TP: The index of the 3' end base that closes the loop.
'-----------------------------------------------------------------------------
Dim Energies#()
Dim NumBases, StemLength, S&

'Check whether this pair can close the loop.
RNA_Hairpin = 1000#
Select Case Seq(FP)
Case 0
    If Seq(TP) <> 3 Then Exit Function
Case 1
    If Seq(TP) <> 2 Then Exit Function
Case 2
    If (Seq(TP) <> 1 And Seq(TP) <> 3) Then Exit Function
Case 3
    If (Seq(TP) <> 0 And Seq(TP) <> 2) Then Exit Function
End Select

'Calculate sequence size.
NumBases = UBound(Seq) - LBound(Seq) + 1

'Setup to store all possible energies.
```

```
ReDim Energies(UBound(Seq) - LBound(Seq) + 1)

'Start the energy calculations with the loop.
Energies(0) = RNAHP_LoopG(2, TP - FP - 2)

'Add the stacking interaction of the first mismatch in the loop.
Energies(0) = Energies(0) + RNAHP_TStackG(Seq(FP), Seq(FP + 1), Seq(TP), Seq(TP
- 1))

'Loop over stem members
S = 0
Do While True
    FP = FP - 1
    TP = TP + 1

    'Check that there are still bases to process.
    If (FP >= 0) And (TP < NumBases) Then

        'Check that we are still in the helix.
        Select Case Seq(FP)
        Case 0
            If Seq(TP) <> 3 Then Exit Do
        Case 1
            If Seq(TP) <> 2 Then Exit Do
        Case 2
            If (Seq(TP) <> 1 And Seq(TP) <> 3) Then Exit Do
        Case 3
            If (Seq(TP) <> 0 And Seq(TP) <> 2) Then Exit Do
        End Select

        'Add the next stacking term
        Energies(S + 1) = Energies(S) + RNAHP_StackG(Seq(FP), Seq(FP + 1),
Seq(TP), Seq(TP - 1))

        'Record the energy if the helix breaks here.
        Energies(S) = Energies(S) + RNAHP_DangleG(1, Seq(TP - 1), Seq(FP + 1),
Seq(FP)) + _
            RNAHP_DangleG(0, Seq(TP - 1), Seq(TP), Seq(FP + 1))

        S = S + 1
    Else
        Exit Do
    End If
Loop

'Add dangles, if they exist.
If (FP >= 0) Then Energies(S) = Energies(S) + RNAHP_DangleG(1, Seq(TP - 1),
Seq(FP + 1), Seq(FP))
If (TP < NumBases) Then Energies(S) = Energies(S) + RNAHP_DangleG(0, Seq(TP -
1), Seq(TP), Seq(FP + 1))

'Find the minimum energy.
RNA_Hairpin = 1000#
StemLength = S
For S = 0 To StemLength
    If Energies(S) < RNA_Hairpin Then RNA_Hairpin = Energies(S)
Next S
```

```
End Function


Private Function DNA_Hairpin#(Seq%(), ByVal FP&, ByVal TP&)
'------------------------------------------------------------------------------
'Function
'   Calculate the free energy of a DNA hairpin
'Arguments
'    Seq: The sequence, in numeric representation.
'    FP: The index of the 5'-end base that closes the loop.
'    TP: The index of the 3' end base that closes the loop.
'------------------------------------------------------------------------------
Dim Energies#()
Dim NumBases, StemLength, S&

'Check whether this pair can close the loop.
DNA_Hairpin = 1000#
Select Case Seq(FP)
Case 0
    If Seq(TP) <> 3 Then Exit Function
Case 1
    If Seq(TP) <> 2 Then Exit Function
Case 2
    If (Seq(TP) <> 1 And Seq(TP) <> 3) Then Exit Function
Case 3
    If (Seq(TP) <> 0 And Seq(TP) <> 2) Then Exit Function
End Select

'Calculate sequence size.
NumBases = UBound(Seq) - LBound(Seq) + 1

'Setup to store all possible energies.
ReDim Energies(UBound(Seq) - LBound(Seq) + 1)

'Start the energy calculations with the loop.
Energies(0) = DNAHP_LoopG(2, TP - FP - 2)

'Add the stacking interaction of the first mismatch in the loop.
Energies(0) = Energies(0) + DNAHP_TStackG(Seq(FP), Seq(FP + 1), Seq(TP), Seq(TP
- 1))

'Loop over stem members
S = 0
Do While True
    FP = FP - 1
    TP = TP + 1

    'Check that there are still bases to process.
    If (FP >= 0) And (TP < NumBases) Then

        'Check that we are still in the helix.
        Select Case Seq(FP)
        Case 0
            If Seq(TP) <> 3 Then Exit Do
        Case 1
            If Seq(TP) <> 2 Then Exit Do
        Case 2
            If (Seq(TP) <> 1 And Seq(TP) <> 3) Then Exit Do
        Case 3
            If (Seq(TP) <> 0 And Seq(TP) <> 2) Then Exit Do
        End Select

        'Add the next stacking term
        Energies(S + 1) = Energies(S) + DNAHP_StackG(Seq(FP), Seq(FP + 1),
Seq(TP), Seq(TP - 1))

        'Record the energy if the helix breaks here.
        Energies(S) = Energies(S) + DNAHP_DangleG(1, Seq(TP - 1), Seq(FP + 1),
Seq(FP)) + _
            DNAHP_DangleG(0, Seq(TP - 1), Seq(TP), Seq(FP + 1))

        S = S + 1
    Else
        Exit Do
    End If
Loop

'Add dangles, if they exist.
If (FP >= 0) Then Energies(S) = Energies(S) + DNAHP_DangleG(1, Seq(TP - 1),
Seq(FP + 1), Seq(FP))
If (TP < NumBases) Then Energies(S) = Energies(S) + DNAHP_DangleG(0, Seq(TP -
1), Seq(TP), Seq(FP + 1))

'Find the minimum energy.
DNA_Hairpin = 1000#
StemLength = S
For S = 0 To StemLength
    If Energies(S) < DNA_Hairpin Then DNA_Hairpin = Energies(S)
Next S

End Function


Public Sub InitThermoPars()
'-----------------------------------------------------------------------
'Function:
'   Initialize all the fixed parameters used for Thermo calculations.
'Notes:
'   1. These parameters are essentially constants, but are placed in
'   arrays for ease-of-use.  Thus this routine must be called by
'   Main to initialize everything.
'   2. The nearest-neighbor parameters are accessed by indexing by
'   the first then second base, with A->0, C->1, G->2, TU->3
'   3. The hairpin parameters are loaded from files by this routine,
'   and the entropy matrices filled in.  After this operation, the
'   free energy matrices may be over-written at any time.
'-----------------------------------------------------------------------

Dim FileName$       'files opened for Zuker parameters.
Dim foo&()          'bitbucket.

'Initialize the DNA Duplex enthalpy and entropy.
'These parameters are from SantaLucia et al. Biochemistry, v. 35, pp 3555.
'as found by PkW.  DuplexH is in kcal/mol, DuplexS in cal/mol/deg K.
```

```
DNA_DuplexH(0, 0) = -8.4                    'AA or TT
DNA_DuplexH(0, 1) = -8.6                    'AC or GT
DNA_DuplexH(0, 2) = -6.1                    'AG or CT
DNA_DuplexH(0, 3) = -6.5                    'AT
DNA_DuplexH(1, 0) = -7.4                    'CA or TG
DNA_DuplexH(1, 1) = -6.7                    'CC or GG
DNA_DuplexH(1, 2) = -10.1                   'CG
DNA_DuplexH(1, 3) = DNA_DuplexH(0, 2)       'CT
DNA_DuplexH(2, 0) = -7.7                    'GA or TC
DNA_DuplexH(2, 1) = -11.1                   'GC
DNA_DuplexH(2, 2) = DNA_DuplexH(1, 1)       'GG
DNA_DuplexH(2, 3) = DNA_DuplexH(0, 1)       'GT
DNA_DuplexH(3, 0) = -6.3                    'TA
DNA_DuplexH(3, 1) = DNA_DuplexH(2, 0)       'TC
DNA_DuplexH(3, 2) = DNA_DuplexH(1, 0)       'TG
DNA_DuplexH(3, 3) = DNA_DuplexH(0, 0)       'TT

DNA_EndTAH = 0.4

DNA_DuplexS(0, 0) = -23.6                   'AA or TT
DNA_DuplexS(0, 1) = -23                     'AC or GT
DNA_DuplexS(0, 2) = -16.1                   'AG or CT
DNA_DuplexS(0, 3) = -18.8                   'AT
DNA_DuplexS(1, 0) = -19.3                   'CA or TG
DNA_DuplexS(1, 1) = -15.6                   'CC or GG
DNA_DuplexS(1, 2) = -25.5                   'CG
DNA_DuplexS(1, 3) = DNA_DuplexS(0, 2)       'CT
DNA_DuplexS(2, 0) = -20.3                   'GA or TC
DNA_DuplexS(2, 1) = -28.4                   'GC
DNA_DuplexS(2, 2) = DNA_DuplexS(1, 1)       'GG
DNA_DuplexS(2, 3) = DNA_DuplexS(0, 1)       'GT
DNA_DuplexS(3, 0) = -18.5                   'TA
DNA_DuplexS(3, 1) = DNA_DuplexS(2, 0)       'TC
DNA_DuplexS(3, 2) = DNA_DuplexS(1, 0)       'TG
DNA_DuplexS(3, 3) = DNA_DuplexS(0, 0)       'TT

DNA_InitGCS = -5.9
DNA_InitATS = -9
DNA_SelfS = -1.4

'Initialize the DNA/RNA duplex parameters.
'Parameters found in Sugimoto et al., Biochemistry,
'v. 34, pp. 11,211-11,216 (1995).
'
'Note carefully, these numbers are given for the sequence of
'the DNA strand i.e. we assume that the probe is DNA, the target
'is RNA, and the sequence of the probe is given to the routine.
'For example, the parameter in (0,0) is for dAA/rTT.

DR_DuplexH(0, 0) = -11.5                    'AA
DR_DuplexH(0, 1) = -7.8                     'AC
DR_DuplexH(0, 2) = -7#                      'AG
DR_DuplexH(0, 3) = -8.3                     'AU
DR_DuplexH(1, 0) = -10.4                    'CA
DR_DuplexH(1, 1) = -12.8                    'CC
DR_DuplexH(1, 2) = -16.3                    'CG
DR_DuplexH(1, 3) = -9.1                     'CU
DR_DuplexH(2, 0) = -8.6                     'GA
DR_DuplexH(2, 1) = -8#                    'GC
DR_DuplexH(2, 2) = -9.3                   'GG
DR_DuplexH(2, 3) = -5.9                   'GU
DR_DuplexH(3, 0) = -7.8                   'UA
DR_DuplexH(3, 1) = -5.5                   'UC
DR_DuplexH(3, 2) = -9#                    'UG
DR_DuplexH(3, 3) = -7.8                   'UU

DR_InitH = 1.9

DR_DuplexS(0, 0) = -36.4                  'AA
DR_DuplexS(0, 1) = -21.6                  'AC
DR_DuplexS(0, 2) = -19.7                  'AG
DR_DuplexS(0, 3) = -23.9                  'AU
DR_DuplexS(1, 0) = -28.4                  'CA
DR_DuplexS(1, 1) = -31.9                  'CC
DR_DuplexS(1, 2) = -47.1                  'CG
DR_DuplexS(1, 3) = -23.5                  'CU
DR_DuplexS(2, 0) = -22.9                  'GA
DR_DuplexS(2, 1) = -17.1                  'GC
DR_DuplexS(2, 2) = -23.2                  'GG
DR_DuplexS(2, 3) = -12.3                  'GU
DR_DuplexS(3, 0) = -23.2                  'UA
DR_DuplexS(3, 1) = -13.5                  'UC
DR_DuplexS(3, 2) = -26.1                  'UG
DR_DuplexS(3, 3) = -21.9                  'UU

DR_InitS = -3.9

'Load the Zuker hairpin parameters.
FileName = App.Path & "\stack.datd.pw"
ReadZuker FileName, "Stack", DNAHP_StackG, foo
FileName = App.Path & "\stack.dhd.pw"
ReadZuker FileName, "Stack", DNAHP_StackH, foo
CalcSFromGH 4, DNAHP_StackG, DNAHP_StackH, 37#, DNAHP_StackS

FileName = App.Path & "\tstackh.datd.pw"
ReadZuker FileName, "Stack", DNAHP_TStackG, foo
FileName = App.Path & "\tstackh.dhd.pw"
ReadZuker FileName, "Stack", DNAHP_TStackH, foo
CalcSFromGH 4, DNAHP_TStackG, DNAHP_TStackH, 37#, DNAHP_TStackS

FileName = App.Path & "\dangle.datd.pw"
ReadZuker FileName, "Dangle", DNAHP_DangleG, foo
FileName = App.Path & "\dangle.dhd.pw"
ReadZuker FileName, "Dangle", DNAHP_DangleH, foo
CalcSFromGH 4, DNAHP_DangleG, DNAHP_DangleH, 37#, DNAHP_DangleS

FileName = App.Path & "\loop.datd.pw"
ReadZuker FileName, "Loop", DNAHP_LoopG, ZLoopLengths
FileName = App.Path & "\loop.dhd.pw"
ReadZuker FileName, "Loop", DNAHP_LoopH, ZLoopLengths
CalcSFromGH 2, DNAHP_LoopG, DNAHP_LoopH, 37#, DNAHP_LoopS

FileName = App.Path & "\tloop.datd.pw"
ReadZuker FileName, "TetraLoop", DNAHP_TLoopG, ZTetraLoops
FileName = App.Path & "\tloop.dhd.pw"
ReadZuker FileName, "TetraLoop", DNAHP_TLoopH, ZTetraLoops
```

```
CalcSFromGH 1, DNAHP_TLoopG, DNAHP_TLoopH, 37#, DNAHP_TLoopS

FileName = App.Path & "\stack.dat.pw"
ReadZuker FileName, "Stack", RNAHP_StackG, foo
FileName = App.Path & "\stack.dh.pw"
ReadZuker FileName, "Stack", RNAHP_StackH, foo
CalcSFromGH 4, RNAHP_StackG, RNAHP_StackH, 37#, RNAHP_StackS

FileName = App.Path & "\tstackh.dat.pw"
ReadZuker FileName, "Stack", RNAHP_TStackG, foo
FileName = App.Path & "\tstackh.dh.pw"
ReadZuker FileName, "Stack", RNAHP_TStackH, foo
CalcSFromGH 4, RNAHP_TStackG, RNAHP_TStackH, 37#, RNAHP_TStackS

FileName = App.Path & "\dangle.dat.pw"
ReadZuker FileName, "Dangle", RNAHP_DangleG, foo
FileName = App.Path & "\dangle.dh.pw"
ReadZuker FileName, "Dangle", RNAHP_DangleH, foo
CalcSFromGH 4, RNAHP_DangleG, RNAHP_DangleH, 37#, RNAHP_DangleS

FileName = App.Path & "\loop.dat.pw"
ReadZuker FileName, "Loop", RNAHP_LoopG, ZLoopLengths
FileName = App.Path & "\loop.dh.pw"
ReadZuker FileName, "Loop", RNAHP_LoopH, ZLoopLengths
CalcSFromGH 2, RNAHP_LoopG, RNAHP_LoopH, 37#, RNAHP_LoopS

FileName = App.Path & "\tloop.dat.pw"
ReadZuker FileName, "TetraLoop", RNAHP_TLoopG, ZTetraLoops
FileName = App.Path & "\tloop.dh.pw"
ReadZuker FileName, "TetraLoop", RNAHP_TLoopH, ZTetraLoops
CalcSFromGH 1, RNAHP_TLoopG, RNAHP_TLoopH, 37#, RNAHP_TLoopS

End Sub


Private Sub ReadZuker(FileName$, ZType$, Param#(), FirstCol&())
'-----------------------------------------------------------------------
'Function
'   Read thermodynamic parameters for structure calculations.
'Arguments
'   Filename: The data file.
'   ZType: The type of parameter array (see below).
'   Param: The parameter array to be filled in.
'   FirstCol: The values from the first column.
'Notes
'   1. This routine reads data files originally designed to be read by
'      MFOLD; the files must be edited to place a # sign on lines that
'      do not contain data, and to replace all the "." entries by 0.
'   2. There are several types of parameter files.  Type "stack"
'      gives the parameters for stacking one base pair over another,
'      and thus has 256 entries.  Type "dangle" gives the parameters
'      for dangling a base over a pair, and thus has 64 parameters.
'      The other types are particular to the parameters being
'      presented.
'   3. The FirstCol argument is used only by Loop and TetraLoop types.
'   4. In all these files, the order of bases is ACGU->0,1,2,3
'-----------------------------------------------------------------------
```

```
Dim ZFile&          'parameter file
Dim FileLength&     'length of parameter file
Dim LineLength&     'length of one line
Dim WordLength&     'length of one word
Dim FileStr$        'text of read-in file
Dim LineStr$        'text of one line
Dim WordStr$        'text of one word
Dim A&, B&, X&, Y&, L&, C&, T& 'loop indices
Dim NumSeq#(0 To 3) 'numeric representation of tetraloop sequences

'Open the file, read it in.
ZFile = FreeFile
Open FileName For Binary As #ZFile
FileLength = FileLen(FileName)
FileStr = Input(FileLength, #ZFile)

'Choose how to parse the file.
Select Case ZType

Case "Stack"
'These entries represent stacking of one base pair over another:
'   5'-AX-3'
'   3'-BY-5'.
'In each row, Y varies fast, B varies slow.
'By row, X varies fast, and A varies slowly.
'Index order used is param(A,X,B,Y).
For A = 0 To 3
    For X = 0 To 3
        NextLine FileStr, FileLength, LineStr, LineLength, "#"
        For B = 0 To 3
            For Y = 0 To 3
                NextWord LineStr, LineLength, WordStr, WordLength
                Param(A, X, B, Y) = CDbl(WordStr)
            Next Y
        Next B
    Next X
Next A

Case "Dangle"
'These entries represent dangles of 3' ends, then the 5' ends:
'   5'-AX-3'
'   3'-B -5' is a 3' dangle
'and
'   5'-A -3'
'   3'-BY-5' is a 5' dangle.
'In each row, X or Y varies fast, and B varies slowly.
'By row, A varies.
'Index order used is param(0,A,X,B) for 3', and param(1,A,B,Y) for the 5'
dangles.
For A = 0 To 3
    NextLine FileStr, FileLength, LineStr, LineLength, "#"
    For B = 0 To 3
        For X = 0 To 3
            NextWord LineStr, LineLength, WordStr, WordLength
            Param(0, A, X, B) = CDbl(WordStr)
        Next X
    Next B
```

```
Next A
For A = 0 To 3
    NextLine FileStr, FileLength, LineStr, LineLength, "#"
    For B = 0 To 3
        For Y = 0 To 3
            NextWord LineStr, LineLength, WordStr, WordLength
            Param(1, A, B, Y) = CDbl(WordStr)
        Next Y
    Next B
Next A

Case "Loop"
'These entries represent internal=0, bulge=1, and hairpin=2 loops.
'By row, looplength L varies.
'Index order used is param(looptype,L)
'FirstCol is filled from the first column, representing loop lengths.
For L = 0 To NumZukerLoops - 1
    NextLine FileStr, FileLength, LineStr, LineLength, "#"
    NextWord LineStr, LineLength, WordStr, WordLength
    FirstCol(L) = CLng(WordStr)
    For T = 0 To 2
        NextWord LineStr, LineLength, WordStr, WordLength
        Param(T, L) = CDbl(WordStr)
    Next T
Next L

Case "TetraLoop"
'These represent especially stable tetraloops.
'By row, sequence of the tetraloop varies.
'Firstcol is filled with a quaternary index of the sequence.
For L = 0 To NumTetraLoops - 1
    NextLine FileStr, FileLength, LineStr, LineLength, "#"
    NextWord LineStr, LineLength, WordStr, WordLength
    DNA_Str2Num WordStr, NumSeq
    FirstCol(L) = 0
    For C = 0 To 3
        FirstCol(L) = FirstCol(L) * 4 + NumSeq(C)
    Next
    NextWord LineStr, LineLength, WordStr, WordLength
    Param(L) = CDbl(WordStr)
Next L

End Select
Close ZFile
End Sub

Function DR_CalcDeltaH#(ByVal Seq$)
'-----------------------------------------------------------------------------
'Function
'   Calculate the association enthalpy for a given RNA sequence
'   with its perfect W-C DNA complement.
'Arguments
'   Seq: The RNA sequence.
'Returns
'   Enthalpy of association, in kcal/mole.
'Notes
'   1. Standard conditions (1M NA+) is assumed.
'History
```

```
'   29-Jul-1997: From PkW's routine of the same name.  PW.
'---------------------------------------------------------------------------

Dim Length&          'length of the sequence
Dim B&               'base index for traversing the sequence
Dim NumSeq%()        'numerical representation of the sequence

'Lower case, please.
Seq = LCase(Seq)

'Create the numeric representation.
Length = Len(Seq)
ReDim NumSeq(0 To Length - 1)
DNA_Str2Num Seq, NumSeq

'Initialize with initiation deltaH
DR_CalcDeltaH = DR_InitH

'Sum the nearest neighbor values along the strand.
For B = 1 To Length - 1
    DR_CalcDeltaH = DR_CalcDeltaH + DR_DuplexH(NumSeq(B - 1), NumSeq(B))
Next B

'Convert to cal.
DR_CalcDeltaH = DR_CalcDeltaH * 1000#

End Function


Function DNA_CalcDeltaH#(ByVal Seq$)
'---------------------------------------------------------------------------
'Function
'   Calculate the association enthalpy for a DNA sequence
'   with its perfect W-C complement.
'Arguments
'   Seq: The sequence.
'Returns
'   Enthalpy of association, in kcal/mole.
'Notes
'   1. Standard conditions (1M NA+) is assumed.
'History
'   29-Jul-1997: From PkW's routine of the same name.  PW.
'---------------------------------------------------------------------------

Dim Length&          'length of the sequence
Dim B&               'base index for traversing the sequence
Dim NumSeq%()        'numerical representation of the sequence

'Lower case, please.
Seq = LCase(Seq)

'Create the numeric representation.
Length = Len(Seq)
ReDim NumSeq(0 To Length - 1)
DNA_Str2Num Seq, NumSeq

'Initialize with AT end correction.
If NumSeq(0) = 3 Then DNA_CalcDeltaH = DNA_CalcDeltaH + DNA_EndTAH
```

```
If NumSeq(Length - 1) = 0 Then DNA_CalcDeltaH = DNA_CalcDeltaH + DNA_EndTAH

'Sum the nearest neighbor values along the strand.
For B = 1 To Length - 1
    DNA_CalcDeltaH = DNA_CalcDeltaH + DNA_DuplexH(NumSeq(B - 1), NumSeq(B))
Next B

'Convert to cal.
DNA_CalcDeltaH = DNA_CalcDeltaH * 1000#

End Function


Function DNA_CalcDeltaS#(ByVal Seq$)
'---------------------------------------------------------------------------
'Function
'   Calculate the association entropy for a sequence
'   with its perfect W-C complement.
'Arguments
'   Seq: The sequence
'Returns:
'   Entropy of association, in cal/mole/deg K.
'Notes
'   1. Parameters are derived from SantaLucia et al., Biochemistry,
'      v. 35, pp. 3555-3562 (1996).
'   2. Standard conditions (1M NA+) is assumed.
'History
'   29-Jul-1997: From PkW's routine of the same name.  PW.
'---------------------------------------------------------------------------

Dim Length&          'length of the sequence
Dim B&               'base index for traversing the sequence
Dim NumSeq%()        'numerical representation of the sequence

'Lower case, please.
Seq = LCase(Seq)

'Create the numeric representation.
Length = Len(Seq)
ReDim NumSeq(Length - 1)
DNA_Str2Num Seq, NumSeq

'Initialize with self-symmetry correction.
If Seq = DNA_RevComp(Seq) Then DNA_CalcDeltaS = DNA_CalcDeltaS + DNA_SelfS

'add initiation term
If InStr(Seq, "c") Or InStr(Seq, "g") Then
    DNA_CalcDeltaS = DNA_CalcDeltaS + DNA_InitGCS
Else
    DNA_CalcDeltaS = DNA_CalcDeltaS + DNA_InitATS
End If

'Sum the nearest neighbor values along the strand
For B = 1 To Length - 1
    DNA_CalcDeltaS = DNA_CalcDeltaS + DNA_DuplexS(NumSeq(B - 1), NumSeq(B))
Next B
```

```
End Function


Function DR_CalcDeltaS#(ByVal Seq$)
' ---------------------------------------------------------------------------
'Function
'   Calculate the association entropy for a given RNA sequence
'   with its perfect W-C DNA complement.
'Arguments
'   Seq: The sequence
'Returns:
'   Entropy of association, in cal/mole/deg K.
'Notes
'   1. Standard conditions (1M NA+) is assumed.
'History
'   29-Jul-1997: From PkW's routine of the same name.  PW.
' ---------------------------------------------------------------------------

Dim Length&          'length of the sequence
Dim B&               'base index for traversing the sequence
Dim NumSeq%()        'numerical representation of the sequence

'Lower case, please.
Seq = LCase(Seq)

'Create the numeric representation.
Length = Len(Seq)
ReDim NumSeq(0 To Length - 1)
DNA_Str2Num Seq, NumSeq

'Begin with initiation term.
DR_CalcDeltaS = DR_InitS

'Sum the nearest neighbor values along the strand.
For B = 1 To Length - 1
    DR_CalcDeltaS = DR_CalcDeltaS + DR_DuplexS(NumSeq(B - 1), NumSeq(B))
Next B

End Function

Public Sub DNA_CalcAllTM(Seq$(), tmp As cTMPars, TM#())
'----------------------------------------------------------------------------
'Function
'   Calculate the melting temperature of an array of oligos with their
'   perfect W-C complements.
'Arguments
'   Seq: The sequences
'   TMP: An instance of the parameter class for TM calculations.
'Returns
'   Melting temperature, in deg. C.
'Notes
'   1. Concentration is used as is, i.e. assuming the complement
'   is present at much lower concentration.
'History
'   29-Jul-1997: From PkW's routine of the same name.  PW.
' ---------------------------------------------------------------------------
```

```
On Error GoTo E

Dim NumSeqs&         'number of sequences we are working with
Dim S&               'index

'Determine the number of probes we are calculating TM for.
NumSeqs = UBound(Seq) + 1

'Calculate melting points
For S = 0 To NumSeqs - 1
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    TM(S) = DNA_CalcTM(Seq(S), tmp.Conc)
Next S
Exit Sub

E: Debug.Print "Error in DNA_CalcAllTM"
Err.Raise Err.Number, , Err.Description
End Sub

Public Sub DNA_CalcClamp(Seq$(), TClamp As cClampPars, Clamp#())
'----------------------------------------------------------------------------
'Function
'   Calculate the melting temperature of the Clamp of a probe
'   to its perfect W-C complement.
'Arguments
'   Seq: The sequences
'   TClamp: An instance of the parameter class for Clamp calculations.
'Returns
'   Melting temperature of tightest clamp, in deg. C.
'Notes
'   1. Concentration is used as is, i.e. assuming the complement
'   is present at much lower concentration.
' ---------------------------------------------------------------------------
On Error GoTo E

Dim NumSeqs&         'number of sequences we are working with
Dim S&, SS&          'indices
Dim SubSeq$          'subsequence
Dim BestTM#, ThisTM# 'current most stable clamp

'Determine the number of probes we are calculating Clamp for.
NumSeqs = UBound(Seq) + 1

'Calculate melting points.
For S = 0 To NumSeqs - 1
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    BestTM = 0
    For SS = TClamp.FiveP + 1 To Len(Seq(S)) - TClamp.ThreeP - TClamp.Length + 1
        SubSeq = Mid(Seq(S), SS, TClamp.Length)
        ThisTM = DNA_CalcTM(SubSeq, TClamp.Conc) + 273.15
        If ThisTM > BestTM Then BestTM = ThisTM
    Next SS
    Clamp(S) = BestTM
Next S
Exit Sub
E: Debug.Print "Error in DNA_CalcClamp"
Err.Raise Err.Number, , Err.Description
End Sub
```

```
Public Sub DR_CalcClamp(Seq$(), TClamp As cClampPars, Clamp#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the melting temperature of the Clamp of a probe
'   to its perfect RNA W-C complement.
'Arguments
'   Seq: The sequences
'   TClamp: An instance of the parameter class for Clamp calculations.
'Returns
'   Melting temperature of tightest clamp, in deg. C.
'Notes
'   1. Concentration is used as is, i.e. assuming the complement
'   is present at much lower concentration.
' ----------------------------------------------------------------------------
On Error GoTo E

Dim NumSeqs&          'number of sequences we are working with
Dim S&, SS&           'indices
Dim SubSeq$           'subsequence
Dim BestTM#, ThisTM#  'current most stable clamp

'Determine the number of probes we are calculating Clamp for.
NumSeqs = UBound(Seq) + 1

'Calculate melting points.
For S = 0 To NumSeqs - 1
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    BestTM = 0
    For SS = TClamp.FiveP + 1 To Len(Seq(S)) - TClamp.ThreeP - TClamp.Length + 1
        SubSeq = Mid(Seq, SS, TClamp.Length)
        ThisTM = DR_CalcTM(SubSeq, TClamp.Conc) + 273.15
        If ThisTM > BestTM Then BestTM = ThisTM
    Next SS
    Clamp(S) = BestTM
Next S
Exit Sub
E: Debug.Print "Error in DR_CalcClamp"
Err.Raise Err.Number, , Err.Description
End Sub

Public Sub DNA_CalcdGH(Seq$(), dGHP As cDGHPars, dGH#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the hairpin dGs of an array of oligos.
'Arguments
'   Seq: The sequences.
'   dGHP: An instance of the parameter class for dGH calculations.
'   dGH: The hairpin dGs.
' ----------------------------------------------------------------------------
On Error GoTo E

Dim NumSeq%()   'numeric representation of the sequence
Dim S&          'index

'Recalculate all G parameter matrices at current temperature.
```

```
CalcGFromHS 4, DNAHP_StackH, DNAHP_StackS, dGHP.T, DNAHP_StackG
CalcGFromHS 4, DNAHP_TStackH, DNAHP_TStackS, dGHP.T, DNAHP_TStackG
CalcGFromHS 4, DNAHP_DangleH, DNAHP_DangleS, dGHP.T, DNAHP_DangleG
CalcGFromHS 2, DNAHP_LoopH, DNAHP_LoopS, dGHP.T, DNAHP_LoopG
CalcGFromHS 1, DNAHP_TLoopH, DNAHP_TLoopS, dGHP.T, DNAHP_TLoopG

'Calculate dGs.
For S = 0 To UBound(Seq)
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    ReDim NumSeq(0 To Len(Seq(S)) - 1)
    DNA_Str2Num Seq(S), NumSeq
    dGH(S) = DNA_BestHairpin(NumSeq)
Next S
Exit Sub
E: Debug.Print "Error in DNA_CalcGH"
Err.Raise Err.Number, , Err.Description
End Sub

Public Sub DNA_CalcdGD(Seq$(), dGDP As cdGDPars, dGD#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the duplex dGs of an array of oligos.
'Arguments
'   Seq: The sequences.
'   dGDP: An instance of the parameter class for dGD calculations.
'   dGD: The hairpin dGs.
' ----------------------------------------------------------------------------
On Error GoTo E

Dim NumSeq%()   'numeric representation of the sequence
Dim S&         .'index

'Calculate dGs.
For S = 0 To UBound(Seq)
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    dGD(S) = (DNA_CalcDeltaH(Seq(S)) - (dGDP.T + 273.15) *
DNA_CalcDeltaS(Seq(S))) / 1000#
Next S
Exit Sub
E: Debug.Print "Error in DNA_CalcGD"
Err.Raise Err.Number, , Err.Description
End Sub

Public Sub DR_CalcdGD(Seq$(), dGDP As cdGDPars, dGD#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the duplex dGs of an array of oligos.
'Arguments
'   Seq: The sequences.
'   dGDP: An instance of the parameter class for dGD calculations.
'   dGD: The hairpin dGs.
' ----------------------------------------------------------------------------
On Error GoTo E

Dim NumSeq%()   'numeric representation of the sequence
Dim S&          'inex

'Calculate dGs.
```

```
For S = 0 To UBound(Seq)
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    dGD(S) = (DR_CalcDeltaH(Seq(S)) - (dGDP.T + 273.15) * DR_CalcDeltaS(Seq(S)))
/ 1000#
Next S
Exit Sub
E: Debug.Print "Error in DNA_CalcGD"
Err.Raise Err.Number, , Err.Description
End Sub


Public Sub DNA_CalcdGM(Seq$(), dGMP As cdGMPars, dGM#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the MFold dGs of an array of oligos.
'Arguments
'   Seq: The sequences.
'   dGMP: An instance of the parameter class for dGM calculations.
'   dGM: The hairpin dGs.
' ----------------------------------------------------------------------------
On Error GoTo E

Dim S&          'index

'Calculate dGs.
For S = 0 To UBound(Seq)
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    dGM(S) = frmMain.Mfoldx.mfold(Seq(S), Val(dGMP.T), True)
Next S
Exit Sub

E: Debug.Print "Error in DNA_CalcdGM."
Err.Raise Err.Number, , Err.Description
End Sub


Public Sub RNA_CalcdGH(Seq$(), dGHP As cDGHPars, dGH#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the hairpin dGs of an array of oligos.
'Arguments
'   Seq: The sequences.
'   dGHP: An instance of the parameter class for dGH calculations.
'   dGH: The hairpin dGs.
' ----------------------------------------------------------------------------
On Error GoTo E

Dim NumSeq%()   'numeric representation of the sequence
Dim S&          'index

'Recalculate all G parameter matrices at current temperature.
CalcGFromHS 4, RNAHP_StackH, RNAHP_StackS, dGHP.T, RNAHP_StackG
CalcGFromHS 4, RNAHP_TStackH, RNAHP_TStackS, dGHP.T, RNAHP_TStackG
CalcGFromHS 4, RNAHP_DangleH, RNAHP_DangleS, dGHP.T, RNAHP_DangleG
CalcGFromHS 2, RNAHP_LoopH, RNAHP_LoopS, dGHP.T, RNAHP_LoopG
CalcGFromHS 1, RNAHP_TLoopH, RNAHP_TLoopS, dGHP.T, RNAHP_TLoopG

'Calculate dGs.
```

```
For S = 0 To UBound(Seq)
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    ReDim NumSeq(0 To Len(Seq(S)) - 1)
    DNA_Str2Num Seq(S), NumSeq
    dGH(S) = RNA_BestHairpin(NumSeq)
Next S
Exit Sub
E: Debug.Print "Error in RNA_CalcdGH"
Err.Raise Err.Number, , Err.Description
End Sub

Public Sub RNA_CalcdGM(Seq$(), dGMP As cdGMPars, dGM#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the MFold dGs of an array of oligos.
'Arguments
'   Seq: The sequences.
'   dGMP: An instance of the parameter class for dGM calculations.
'   dGM: The MFold dGs.
' ----------------------------------------------------------------------------
On Error GoTo E

Dim S&          'index

'Calculate dGs.
For S = 0 To UBound(Seq)
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    dGM(S) = frmMain.Mfoldx.mfold(Seq(S), Val(dGMP.T), False)
Next S
Exit Sub
E: Debug.Print "Error in RNA_CalcGM" .
Err.Raise Err.Number, , Err.Description
End Sub


Public Sub DR_CalcAllTM(Seq$(), tmp As cTMPars, TM#())
'-----------------------------------------------------------------------------
'Function
'   Calculate the melting temperature of an array of DNA probes with their
'   perfect W-C RNA target complements.
'Arguments
'   Seq: The sequences.
'   TMP: An instance of the parameter class for TM calculations.
'Returns
'   Melting temperature, in deg. C.
'Notes
'   1. Concentration is used as is, i.e. assuming the complement
'   is present at much lower concentration.
'History
'   29-Jul-1997: From PkW's routine of the same name.  PW.
' ----------------------------------------------------------------------------
On Error GoTo E
Dim NumSeqs&            'number of sequences we are working with
Dim S&                  'index

'Determine the number of probes we are calculating TM for.
NumSeqs = UBound(Seq) - LBound(Seq) + 1
```

```
'Calculate melting points
For S = 0 To NumSeqs - 1
    If S - Progress.StopAt = 0 Then Progress.CheckProgress S
    TM(S) = DR_CalcTM(Seq(S), tmp.Conc)
Next S
Exit Sub
E: Debug.Print "Error in DR_CalcAllTM"
Err.Raise Err.Number, , Err.Description
End Sub

Public Function DR_CalcTM#(Seq$, Conc#)
'---------------------------------------------------------------------------
'Function
'   Calculate the TM of one DNA/RNA duplex.  Sequence given is DNA.
'---------------------------------------------------------------------------
DR_CalcTM = DR_CalcDeltaH(Seq) / (DR_CalcDeltaS(Seq) + RGas * Log(Conc)) -
273.15
End Function

Public Function DNA_CalcTM#(Seq$, Conc#)
'---------------------------------------------------------------------------
'Function
'   Calculate the TM of one DNA/DNA duplex.
'---------------------------------------------------------------------------
DNA_CalcTM = DNA_CalcDeltaH(Seq) / (DNA_CalcDeltaS(Seq) + RGas * Log(Conc)) -
273.15
End Function
```

```
Attribute VB_Name = "Utilities"
Option Explicit

'A quicksort routine for doubles, based on NR code.
Private Declare Function vb5sort2 Lib "vb5nrdll.dll" _
    (ByVal N&, ByRef Vector#, ByRef Index&) As Long

Public Sub CalcRun(Pos&(), RP As cRunPars, Run&())
'---------------------------------------------------------------------------
'Function
'   Calculate the runs in a set of positions.
'Arguments
'   Pos: The positions.
'   RP: The run parameters.
'   Run: The returned run.
'Method
'   Begin by looping over all positions, extending down over all consecutive
'   entries, then up over all consecutive entries.  Then mark in the active
'   position the length of run found.  Next, eliminate all runs that are
'   too short.  Finally, for each run, mark the requested number of members
'   of the run, at the requested spacing, as being members.  If the run is
'   shorter than can accomodate the requested number of members, fill in as
'   many as possible, while still preserving the requested spacing.
'Notes
'   1. Run must be allocated and sized correctly by the caller.
'---------------------------------------------------------------------------
Dim P&              'index
Dim EndUp&          'end of the run, counting up
Dim EndDown&        'end of the run, counting down
```

```
Dim StepDir&        'current step direction
Dim M&              'index

'Loop over positions.
For P = 0 To UBound(Pos)
    EndDown = P
    Do While ((P - EndDown) = (Pos(P) - Pos(EndDown)))
        EndDown = EndDown - 1
        If EndDown = -1 Then Exit Do
    Loop
    EndDown = EndDown + 1
    EndUp = P
    Do While ((EndUp - P) = (Pos(EndUp) - Pos(P)))
        EndUp = EndUp + 1
        If EndUp = UBound(Pos) + 1 Then Exit Do
    Loop
    EndUp = EndUp - 1
    Run(P) = EndUp - EndDown + 1
Next P

'Remove runs that are too short.
For P = 0 To UBound(Run)
    If Run(P) < RP.Min Then Run(P) = 0
Next P

'Pick out requested elements from each run.
P = 0
Do While P < UBound(Run)

    'Find next run.
    Do While Run(P) = 0 And P < UBound(Run)
        P = P + 1
    Loop

    'Record the ends, step to the middle (to the right of middle for even
lengths).
    EndDown = P
    EndUp = P + Run(P) - 1
    P = Fix((EndDown + EndUp + 1) / 2)

    'Mark elements, stepping down from the middle first.  This ensures that all
    'elements get marked for even lengths, spacing=1.
    StepDir = -1
    For M = 1 To RP.Num
        Run(P) = -Run(P)
        P = P + StepDir * M * RP.Spacing
        If (P < EndDown Or P > EndUp) Then Exit For
        StepDir = StepDir * -1
    Next M

    'Move over this run.
    P = EndUp + 1
Loop

'Convert marked elements back to run length.
For P = 0 To UBound(Run)
    If Run(P) > 0 Then Run(P) = 0
    If Run(P) < 0 Then Run(P) = -Run(P)
```

```
Next P

End Sub

Public Function IsGoodRS(RS As Recordset) As Boolean
'-----------------------------------------------------------------------------
'Function
'   Check whether it is possible to access the records of
'   recordset connected to the sequence, probeset, or probes table.
'   No current record need exist for this function to return True.
'-----------------------------------------------------------------------------
On Error GoTo Err
IsGoodRS = False
If IsNull(RS) Then Exit Function
If (RS.EOF = True And RS.BOF = True) Then Exit Function
IsGoodRS = True
Err:
End Function

Public Function NumRecords&(RS As Recordset)
'-----------------------------------------------------------------------------
'Function
'   Count the number of records in the recordset.
'Notes
'   A side effect of this routine is to position the recordset at the
'   first record.
'-----------------------------------------------------------------------------
RS.MoveLast
RS.MoveFirst
NumRecords = RS.RecordCount
End Function

Public Sub UnPackSequence(ByRef SeqStr As String, ByRef NumStr As String)
'SeqStr is assumed to hold a packed sequence.
Dim NewSeqStr$
Dim Bases&, I&

NumStr = ""

Bases = 1
Do While Len(SeqStr) - Bases > 60
    NumStr = NumStr & Format(Format(Bases, "#####"), "@@@@@") & vbCrLf
    For I = Bases To Bases + 59
        NewSeqStr = NewSeqStr & Mid(SeqStr, I, 1)
        If (I Mod 10 = 0) Then NewSeqStr = NewSeqStr & " "
    Next I
    Bases = Bases + 60
    NewSeqStr = NewSeqStr + vbCrLf
Loop
If Bases <> Len(SeqStr) Then
    NumStr = NumStr & Format(Format(Bases, "#####"), "@@@@@")
    For I = Bases To Len(SeqStr)
        NewSeqStr = NewSeqStr & Mid(SeqStr, I, 1)
        If (I Mod 10 = 0) Then NewSeqStr = NewSeqStr & " "
    Next I
End If
SeqStr = NewSeqStr
End Sub
```

```
Public Function StrField$(Str$, lineLen&, ByVal FieldLen&, Optional Extra&)
'Returns a leading field of a string, and shortens the string
'by the fieldlength + extra.
'If the line is too short, as much of the requested field as possible
'will be returned.

If FieldLen > lineLen Then FieldLen = lineLen
StrField = Trim(Left(Str, FieldLen))
lineLen = lineLen - FieldLen
If Not IsMissing(Extra) Then
    lineLen = lineLen - Extra
    If lineLen < 0 Then lineLen = 0
End If
Str = Right(Str, lineLen)
End Function

Public Sub NextWord(Str$, StrLen&, Word$, WordLen&)
'Strips leading spaces, copies leading word, reduces linelength
Do While ((Mid(Str, 1, 1) = " ") And (StrLen > 0))
    StrLen = StrLen - 1
    Str = Right(Str, StrLen)
Loop
If InStr(Str, " ") <> 0 Then
    WordLen = InStr(Str, " ") - 1
Else
    WordLen = StrLen
End If
If WordLen <> 0 Then
    Word = Mid(Str, 1, WordLen)
    StrLen =. StrLen - WordLen
    Str = Right(Str, StrLen)
Else
    Word = ""
    Str = ""
    StrLen = 0
    WordLen = 0
End If
End Sub

Public Sub NextLine(FileStr$, FileLen&, LineStr$, lineLen&, CommentChar$)
'Return the next line that doesn't begin with CommentChar
lineLen = 0
Do While (lineLen = 0 And FileLen > 0)
    lineLen = InStr(FileStr, vbCrLf) - 1
    LineStr = Left(FileStr, lineLen)
    FileLen = FileLen - lineLen - 2
    FileStr = Right(FileStr, FileLen)
    If (Mid$(LineStr, 1, 1) = CommentChar) Then lineLen = 0
Loop
End Sub

Public Sub QuickSort(X#(), Index&(), LB&, UB&)
'Quicksorts the array X into ascending order,
'simultaneously sorting I to provide a sort index.
Dim N&, foo&
N = UB - LB + 1
foo = vb5sort2(N, X(LB), Index(LB))
```

```
End Sub

Public Function BinarySearch&(Vector#(), Value#, U&, L&)
'--------------------------------------------------------------------------------
'Function
'   Find the index of a value in a sorted array.
'--------------------------------------------------------------------------------
Do While U <> L
    BinarySearch = (U + L) / 2
    If Vector(BinarySearch) = Value Then Exit Function
    If Vector(BinarySearch) > Value Then
        U = BinarySearch
    Else
        L = BinarySearch
    End If
Loop
End Function

Public Function PackSequence$(SeqStr$)
Dim NewStr$, Ch$
Dim I&
For I = 1 To Len(SeqStr)
    Ch = Mid$(SeqStr, I, 1)
    If InStr("ACGTUacgtu", Ch) Then PackSequence = PackSequence & Ch
Next I
End Function
```